Span-program-based quantum algorithm for evaluating formulas

Ben W. Reichardt* Robert Špalek†

Abstract

We give a quantum algorithm for evaluating formulas over an extended gate set, including all two- and three-bit binary gates (e.g., NAND, 3-majority). The algorithm is optimal on read-once formulas for which each gate's inputs are balanced in a certain sense.

The main new tool is a correspondence between a classical linear-algebraic model of computation, "span programs," and weighted bipartite graphs. A span program's evaluation corresponds to an eigenvalue-zero eigenvector of the associated graph. A quantum computer can therefore evaluate the span program by applying spectral estimation to the graph.

For example, the classical complexity of evaluating the balanced ternary majority formula is unknown, and the natural generalization of randomized alpha-beta pruning is known to be suboptimal. In contrast, our algorithm generalizes the optimal quantum AND-OR formula evaluation algorithm and is optimal for evaluating the balanced ternary majority formula.

1 Introduction

A formula φ on gate set $\mathcal{S}$ and of size N is a tree with N leaves, such that each internal node is a gate from $\mathcal{S}$ on its children. The read-once formula evaluation problem is to evaluate $\varphi(x)$ given oracle access to the input string $x = x_1 x_2 \dots x_N$. An optimal, $O(\sqrt{N})$-query quantum algorithm is known to evaluate "approximately balanced" formulas over the gates $\mathcal{S} = \{\text{AND, OR, NOT}\}$ [ACR+07]. We extend the gate set $\mathcal{S}$. We develop an optimal quantum algorithm for evaluating balanced, read-once formulas over a gate set $\mathcal{S}$ that includes arbitrary three-bit gates, as well as bounded fan-in EQUAL gates and bounded-size {AND, OR, NOT, PARITY} formulas considered as single gates. The correct notion of "balanced" for a formula including different kinds of gates turns out to be "adversary-balanced," meaning that the inputs to a gate must have exactly equal adversary lower bounds. The definition of "adversary-balanced" formulas also includes as a special case layered formulas in which all gates at a given depth from the root are of the same type.

The idea of our algorithm is to consider a weighted graph $G(\varphi)$ obtained by replacing each gate of the formula φ with a small gadget subgraph, and possibly also duplicating subformulas. Figure 1 has several examples. We relate the evaluation of φ to the presence or absence of small-eigenvalue eigenvectors of the weighted adjacency matrix $A_{G(\varphi)}$ that are supported on the root vertex of $G(\varphi)$. The quantum algorithm runs spectral estimation to either detect these eigenvectors or not, and therefore to evaluate φ.

As a special case, for example, our algorithm implies:

Theorem 1.1. *A balanced ternary majority* (MAJ_3) *formula of depth d, on $N = 3^d$ inputs, can be evaluated by a quantum algorithm with bounded error using $O(2^d)$ oracle queries, which is optimal.*

The classical complexity of evaluating this formula is known only to lie between $\Omega((7/3)^d)$ and $o((8/3)^d)$, and the previous best quantum algorithm, from [ACR+07], used $O(\sqrt{5}^d)$ queries.

*School of Computer Science and Institute for Quantum Computing, University of Waterloo. Part of the work conducted while at the Institute for Quantum Information, California Institute of Technology, supported by NSF Grants CCF-0524828 and PHY-0456720, and by ARO Grant W911NF-05-1-0294. Email: breic@uwaterloo.ca

†Google, Inc. Part of the work conducted while at the University of California, Berkeley, supported by NSF Grant CCF-0524837 and ARO Grant DAAD 19-03-1-0082. Email: spalek@google.com

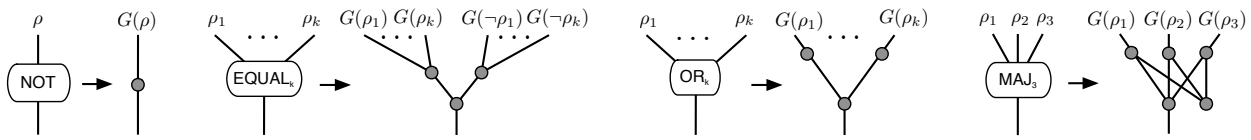


Figure 1: To convert a formula φ to the corresponding graph $G(\varphi)$, we recursively apply substitution rules starting at the root to convert each gate into a gadget subgraph. Some of the rules are shown here, except with the edge weights not indicated. The dangling edges at the top and bottom of each gadget are the input edges and output edge, respectively. To compose two gates, the output edge of one is identified with an input edge of the next (see Figure 4).

The graph gadgets themselves are derived from "span programs" [KW93]. Span programs have been used in classical complexity theory to prove lower bounds on formula size [KW93, BGW99] and monotone span programs are related to linear secret-sharing schemes [BGP96]. (Most, though not all [ABO99], applications are over finite fields, whereas we use the definition over $\mathbf{C}$.) We will only use compositions of constant-size span programs, but it is interesting to speculate that larger span programs could directly give useful new quantum algorithms.

Classical and quantum background The formula evaluation problem has been well-studied in the classical computer model. Classically, the case $\mathcal{S} = \{\text{NAND}\}$ is best understood. A formula with only NAND gates is equivalent to one with alternating levels of AND and OR gates, a so-called "AND-OR formula," also known as a two-player game tree. One can compute the value of a balanced binary AND-OR formula with zero error in expected time $O(N^{\log_2[(1+\sqrt{33})/4]}) = O(N^{0.754})$ [Sni85, SW86], and this is optimal even for bounded-error algorithms [San95]. However, the complexity of evaluating balanced AND-OR formulas grows with the degree of the gates. For example, in the extreme case of a single OR gate of degree N, the complexity is $\Theta(N)$. The complexity of evaluating AND-OR formulas that are not "well-balanced" is unknown.

If we allow the use of a quantum computer with coherent oracle access to the input, however, then the situation is much simpler; between $\Omega(\sqrt{N})$ and $N^{\frac{1}{2}+o(1)}$ queries are necessary and sufficient to evaluate *any* {AND, OR, NOT} formula with bounded error. In one extreme case, Grover search [Gro96, Gro02] evaluates an OR gate of degree N using $O(\sqrt{N})$ oracle queries and $O(\sqrt{N}\log\log N)$ time. In the other extreme case, Farhi, Goldstone and Gutmann recently devised a breakthrough algorithm for evaluating the depth-$\log_2 N$ balanced binary AND-OR formula in $O(\sqrt{N})$ time in the unconventional Hamiltonian oracle model [FGG07]. Ambainis [Amb07] improved this to $O(\sqrt{N})$-queries in the standard query model. Childs, Reichardt, Špalek and Zhang [CRŠZ07] gave an $O(\sqrt{N})$-query algorithm for evaluating balanced or "approximately balanced" formulas, and extended the algorithm to arbitrary {AND, OR, NOT} formulas with $N^{\frac{1}{2}+o(1)}$ queries, and also $N^{\frac{1}{2}+o(1)}$ time after a preprocessing step. (Ref. [ACR+07] contains the merged results of [Amb07, CRŠZ07].)

This paper shows other nice features of the formula evaluation problem in the quantum computer model. Classically, with the exception of {NAND}, {NOR} and a few trivial cases like {PARITY}, most gate sets are poorly understood. In 1986, Boppana asked the complexity of evaluating the balanced, depth-d ternary majority (MAJ_3) function [SW86], and today the complexity is only known to lie between $\Omega((7/3)^d)$ and $O((2.6537\ldots)^d)$ [JKS03]. In particular, the naïve generalization of randomized alpha-beta pruning—recursively evaluate two random immediate subformulas and then the third if they disagree—runs in expected time $O((8/3)^d)$ and is suboptimal. This suggests that the balanced ternary majority function is significantly different from the balanced k-ary NAND function, for which randomized alpha-beta pruning is known to be optimal. In contrast, we show that the optimal quantum algorithm of [CRŠZ07] does extend to give an optimal $O(2^d)$-query algorithm for evaluating the balanced ternary majority formula. Moreover, the algorithm also generalizes to a significantly larger gate set $\mathcal{S}$.

Organization We introduce span programs and explain their correspondence to weighted bipartite graphs in Section 2. The correspondence involves considering parts of a span program P as the weighted adjacency matrix for a corresponding graph G_P. We prove that the eigenvalue-zero eigenvectors of this adjacency matrix evaluate P (Theorem 2.5). This theorem provides useful intuition.

We develop a quantitative version of Theorem 2.5 in Section 3. We lower-bound the overlap of the eigenvalue-zero eigenvector with a known starting state. This lower-bound will imply completeness of our quantum algorithm. To show soundness of the algorithm, we also analyze small-eigenvalue eigenvectors in order to prove a spectral gap around zero. Essentially, we solve the eigenvalue equations in terms of the eigenvalue λ, and expand a series around $\lambda = 0$. The results for small-eigenvalue and eigenvalue-zero eigenvectors are closely related, and we unify them using a measure we term "span program witness size." The details of the proofs from this section are in Appendix A.

Section 4 applies the span program framework to the formula evaluation problem. Theorem 4.7 is our general result, an optimal quantum algorithm for evaluating formulas that are over the gate set $\mathcal{S}$ of Definition 4.1, and that are adversary-balanced (Definition 4.5). The proof of Theorem 4.7 has three parts. First, in Section 4.2, we display an optimal span program for each of the gates in $\mathcal{S}$. Second, we compose the span programs for the individual gates to obtain a span program for the full formula φ. This is equivalent to joining together the gadget graphs described in Figure 1 to obtain a graph $G(\varphi)$. We combine the spectral analyses of the individual span programs to analyze the spectrum of $G(\varphi)$ (Theorem 4.16). Finally, this analysis straightforwardly leads to a quantum algorithm based on phase estimation of a quantum walk on $G(\varphi)$, in Section 4.4.

Section 5 concludes with a discussion of some extensions to the algorithm.

2 Span programs and eigenvalue-zero graph eigenvectors

A span program P is a certain linear-algebraic way of specifying a function f_P. For details on span programs applied in classical complexity theory, we can still recommend the original reference [KW93] as well as, e.g., the more recent [GP03].

Definition 2.1 (Span program)**.** *A span program P consists of a nonzero "target" vector t in a vector space over $\mathbf{C}$, together with "grouped input" vectors $\{v_j : j \in J\}$. Each v_j is labeled with a subset X_j of the literals $\{x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}\}$. To P corresponds a boolean function $f_P : \{0,1\}^n \to \{0,1\}$; defined by $f_P(x) = 1$ (i.e., true) if and only if there exists a linear combination $\sum_j a_j v_j = t$ such that $a_j = 0$ if any of the literals in X_j evaluates to zero (i.e., false).*

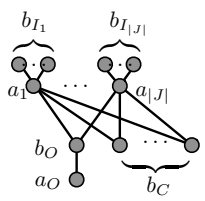
Example 2.2. *For example, the span program*

$$t = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \begin{array}{c} X_J = (\{x_1\} \;\; \{x_2\} \quad \{x_3\} \;\;) \\ v_J = \begin{pmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ 1 & e^{2\pi i/2} & e^{-2\pi i/3} \end{pmatrix} \end{array}$$

computes the MAJ_3 *function. Indeed, at least two of the v_j must have nonzero coefficient in any linear combination equaling the target t. Of course, the second row of $(v_1 \quad v_2 \quad v_3)$ could be any $(\alpha \quad \beta \quad \gamma)$ with α, β, γ distinct and nonzero, and the span program would still compute* MAJ_3*. This specific setting is used to optimize the running time of the quantum algorithm (Claim 4.9).*

In this section, we will show that by viewing a span program P as the weighted adjacency matrix A_{G_P} of a certain graph G_P, the true/false evaluation of P on input x corresponds to the existence or nonexistence of an eigenvalue-zero eigenvector of $A_{G_P}(x)$ supported on a distinguished output node (Theorem 2.5).

In turn, this will imply that writing a span program P for a function f immediately gives a quantum algorithm for evaluating f, or for evaluating formulas including f as a gate (Section 4). The algorithm works by spectral estimation on $A_{G_P}(x)$. Its running time depends on the span program's "witness size" (Section 3). For example, if $f_P(x)$ is true, then the witness size is essentially the shortest squared length of any witness vector $(a_j)_{j \in J}$ in Definition 2.1.



$$A_{G_P} = \begin{array}{c} \begin{array}{cc} a_O & a_J \end{array} \\ \left(\begin{array}{c|c} 1 & A_{OJ} \\ 0 & \\ \vdots & A_{CJ} \\ 0 & \\ \hline 0 & \\ \vdots & A_{IJ} \\ 0 & \end{array}\right) \end{array} \begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{l} b_O \\ \\ \\ \\ \} b_C \\ \\ \} b_I \end{array}$$

Figure 2: The bipartite graph G_P corresponding to span program P (the output edge is (a_O, b_O), while the grouped inputs are $a_1, \ldots, a_{|J|}$).

Remark 2.3. *Let us clarify a few points in Definition 2.1.*

1. *It is convenient, but nonstandard, to allow grouped inputs, i.e., literal subsets X_j possibly with $|X_j| > 1$, instead of just single literals, to label the columns. A grouped input j can be thought of as evaluating the AND of all literals in X_j. A span program P with some $|X_j| > 1$ can be expanded out so that all $|X_j| \leq 1$, without increasing $\sum_j |X_j|$, known as the* size *of P.*

2. *It is sometimes convenient to allow $X_j = \emptyset$. In this case, vector v_j is always available to use in the linear combination; grouped input j evaluates to true always. However, such vectors can be eliminated from P without increasing the size [KW93, Theorem 7].*

3. *By a basis change, one can always adjust the target vector t to $(1, 0, 0, \ldots, 0)$.*

2.1 Span program as an adjacency matrix

A span program P with target vector $t = (1, 0, \ldots, 0)$ corresponds to a certain weighted bipartite graph.

Notation: For an index sequence $H = (h_1, \ldots, h_{|H|})$ and a set of variables $\{a_h\}$, let $a_H = (a_{h_1}, \ldots, a_{h_{|H|}})$. For example, v_J denotes the sequence of grouped input vectors. It will be convenient to define several more index sequences: O ("output"), C ("constraints") and I ("inputs"). Let O and C together index the coordinates of the vector space, with $O = \{1\}$ being the first coordinate, and C the remainder. Let I_j index X_j for each $j \in J$, and let $I = \bigcup_{j \in J} I_j$ a disjoint union so $|I| = \text{size}(P)$.

We will construct a graph G_P on $|I| + |J| + |C| + 2|O|$ vertices. Writing the grouped input vectors out as the columns of a matrix, let $\binom{A_{OJ}}{A_{CJ}} = \sum_{j \in J} |v_j\rangle\langle j|$; A_{OJ} is a $1 \times |J|$ matrix row, and A_{CJ} is a $|C| \times |J|$ matrix. Let $A_{IJ} = \sum_{j \in J, i \in I_j} |i\rangle\langle j|$; A_{IJ} encodes P's grouped inputs. Now consider the bipartite graph G_P of Figure 2, the upper right block of whose weighted Hermitian adjacency matrix is A_{G_P}. (The adjacency matrix is block off-diagonal because the graph is bipartite.) The edges (a_j, b_i) for $j \in J$ and $i \in I_j$ are "input edges," while (a_O, b_O) is the "output edge." The input and output edges all have weight one. The weights of edges (b_O, a_j) for $j \in J$ are given by A_{OJ} (the first coordinates of the grouped input vectors v_J), while the weights of edges (b_c, a_j) for $c \in C, j \in J$ are given by A_{CJ} (the remaining coordinates of v_J).

Example 2.4. *For the* MAJ_3 *span program of Example 2.2, $|C| = 1$, $|I| = |J| = 3$, the graph G_P is shown in Figure 1, and the matrix A_{G_P} is*

$$\left(\begin{array}{c|ccc} 1 & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ 0 & 1 & e^{2\pi i/3} & e^{-2\pi i/3} \\ \hline 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right).$$

2.2 Eigenvalue-zero eigenvectors of the span program adjacency matrix

Theorem 2.5. *For an input $x \in \{0,1\}^n$, define a weighted graph $G_P(x)$ by deleting from G_P the edges (a_j, b_i) if the ith literal in X_j is true. Consider all the eigenvalue-zero eigenvector equations of the weighted adjacency matrix $A_{G_P(x)}$, except for the constraint at a_O. These equations have a solution with support on vertex a_O if and only if $f_P(x) = 1$, and have a solution with support on b_O if and only if $f_P(x) = 0$.*

Proof. Notation: Use a_j, b_i, b_c, a_O, b_O to denote coefficients of a vector on the vertices of G_P. Let $A_{IJ}(x)$ include only edges to false inputs, i.e., $A_{IJ}(x) = \sum_{j \in J, \text{false } i \in I_j} |i\rangle\langle j|$.

The eigenvalue-λ eigenvector equations of $A_{G_P(x)}$ are

$$\lambda b_O = a_O + A_{OJ} a_J \tag{2.1a}$$
$$\lambda b_C = A_{CJ} a_J \tag{2.1b}$$
$$\lambda b_I = A_{IJ}(x) a_J \tag{2.1c}$$
$$\lambda a_O = b_O \tag{2.1d}$$
$$\lambda a_J = A_{OJ}{}^\dagger b_O + A_{CJ}{}^\dagger b_C + A_{IJ}(x)^\dagger b_I \tag{2.1e}$$

At $\lambda = 0$, these equations say that for each vertex, the weighted sum of the adjacent vertices' coefficients must be zero. We are looking for solutions satisfying all these equations except possibly Eq. (2.1d). Since the graph is bipartite, at $\lambda = 0$ the a coefficients do not interact with the b coefficients. In particular, Eqs. (2.1d,e) (resp. 2.1a-c) can always be satisfied by setting the b (resp. a) coefficients to zero.

By scaling, there is a solution with nonzero a_O iff there is a solution with $a_O = -1$. Then Eqs. (2.1a,b) are equivalent to $t = \binom{A_{OJ}}{A_{CJ}} a_J = \sum_j a_j v_j$. Moreover, Eq. (2.1c) implies that a_j can be nonzero only if grouped input j is true. (If X_j includes any false inputs, then $A_{IJ}(x)|j\rangle \neq 0$, so $a_j = 0$.) These conditions are the same as those in Definition 2.1.

Next, we argue that there is a solution of Eq. (2.1e) with $\lambda = 0$ and $b_O = 1$ if and only if $f_P(x) = 0$. Indeed,

$$f_P(x) = 0 \Leftrightarrow t \notin \text{Span}\{v_j : j \text{ true}\} \Leftrightarrow t \notin \text{Range}\left[\binom{A_{OJ}}{A_{CJ}} \Pi\right]$$

where $\Pi = \sum_{\text{true } j} |j\rangle\langle j|$. In turn, this holds iff there is a vector w orthogonal to the range and having inner product one with t—precisely constraint (2.1e) with $w = \binom{b_O}{b_C}$. □

Remark 2.6. *By Theorem 2.5, we can think of the graph G_P as giving a "dual-rail" encoding of the function f_P: there is a $\lambda = 0$ eigenvector of $G_P(x)$ supported on a_O if and only if $f_P(x) = 1$, and there is one supported on b_O iff $f_P(x) = 0$. This justifies calling edge (a_O, b_O) the output edge of G_P.*

2.3 Dual span program

A span program P immediately gives a dual span program, denoted $P^\dagger$, such that $f_{P^\dagger}(x) = \neg f_P(x)$ for all $x \in \{0,1\}^n$. For our purposes, though, it suffices to define a NOT gate graph gadget to allow negation of subformulas.

Definition 2.7 (NOT gate gadget)**.** *Implement a NOT gate $x \mapsto \overline{x}$ as two weight-one edges connected (Figure 1). The edge (a_i, b_i) is the input edge, while (a_O, b_O) is the output edge. The middle vertex $a_i = b_O$ is shared.*

At $\lambda = 0$, the coefficient on a_O is minus that on b_i, and $a_i = b_O$ by definition. Therefore, this gadget complements the dual rail encoding of Theorem 2.5.

The NOT gate gadget of Definition 2.7 can be used to define a dual span program $P^\dagger$ by complementing the output and all inputs with NOT gates, and also complementing all input literals in the sets X_J. Since it is not essential here, we leave the formal definition as an exercise. Alternative constructions of dual programs are given in [CF02, NNP05].

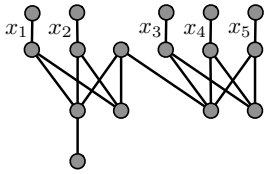


Figure 3: Graph for $\mathrm{MAJ}_3(x_1, x_2, \mathrm{MAJ}_3(x_3, x_4, x_5))$, with input edges labeled by the associated literals.

Example 2.8. *For distinct, nonzero α, β, γ, the span program*

$$X_J = (\ \emptyset\ \emptyset\ \{\overline{x_1}\}\ \{\overline{x_2}\}\ \{\overline{x_3}\})$$

$$t = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad v_J = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & \alpha & 1 & 0 & 0 \\ 1 & \beta & 0 & 1 & 0 \\ 1 & \gamma & 0 & 0 & 1 \end{pmatrix}$$

computes $\neg\,\mathrm{MAJ}_3(x_1, x_2, x_3)$. It was constructed, by adding NOT gate gadgets, as the dual to the span program in Example 2.2, up to choice of weights.

2.4 Span program composition

Definition 2.9 (Composed graph and span program)**.** *Consider span program Q on $\{0,1\}^n$ and programs Q_i, $i \in [n] \equiv \{1, \ldots, n\}$, with corresponding graphs G_Q and G_{Q_i}. The composed graph is defined by identifying the input edges of G_Q with the output edges of copies of the other graphs. If an edge corresponds to input literal x_i, then identify that edge with the output edge of a copy of G_{Q_i}; and if an edge corresponds to $\overline{x_i}$, then insert a NOT gate gadget (i.e., an extra vertex, as in Definition 2.7) before a copy of G_{Q_i}. The composed span program, denoted $Q \circ Q_{[n]}$, is the program corresponding to the composed graph (i.e., $G_{Q \circ Q_{[n]}}$ is the composed graph). Thus $f_{Q \circ Q_{[n]}} = f_Q \circ f_{Q_{[n]}}$.*

Definition 2.10 (Formula graph and span program)**.** *Given span programs for each gate in a formula φ, span program $P(\varphi)$ is defined as their composition according to the formula. Let $G(\varphi)$ be the composed graph, $G(\varphi) = G_{P(\varphi)}$.*

Example 2.11. *For example, the span program*

$$X_J = (\{x_1\}\ \{x_2\}\ \emptyset\ \{x_3\}\ \{x_4\}\ \{x_5\})$$

$$t = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad v_J = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ \alpha & \beta & \gamma & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & \alpha & \beta & \gamma \end{pmatrix}$$

is a composed span program that computes the function $\mathrm{MAJ}_3(x_1, x_2, \mathrm{MAJ}_3(x_3, x_4, x_5))$, provided α, β, γ are distinct and nonzero. (See Example 2.2.) Figure 3 shows the associated composed graph.

Example 2.12 (Duplicating and negating inputs)**.** *To the left in Figure 4 is the composed graph for the formula $\mathrm{MAJ}_3(x_1, x_2, x_3) \oplus x_4 = \mathrm{EQUAL}_2(\mathrm{MAJ}_3(x_{[3]}), \overline{x_4})$, obtained using the substitution rules of Figure 1. (A span program for PARITY will be given in Lemma 4.12.) Note that we are effectively negating some inputs twice, by putting NOT gate gadgets below the negated literals $\overline{x_1}, \overline{x_2}, \overline{x_3}$. This is of course redundant, and is only done to maintain the strict correspondence of graphs to span programs, as in Example 2.8, by keeping the input vertices b_I at odd distances from a_O.*

To the right is the same graph evaluated on input $x = 1100$, i.e., with edges to true literals deleted. Since the formula evaluates to true, Theorem 2.5 promises that there is a $\lambda = 0$ eigenvector supported on a_O. In this case, that eigenvector is unique. It has support on the black vertices.

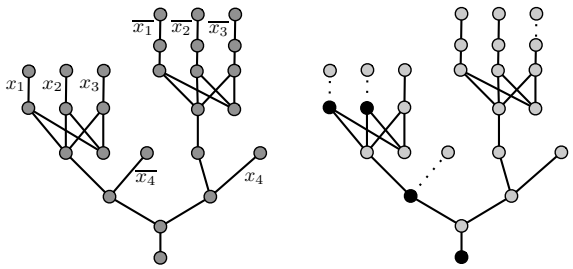


Figure 4: Graph for $\mathrm{MAJ}_3(x_1, x_2, x_3) \oplus x_4$, and its evaluation on input $x = 1100$.

3 Span program witness size

In Section 2, we established that after converting a formula φ into a weighted graph $G(\varphi)$, by replacing each gate with a gadget subgraph coming from a span program, the eigenvalue-zero eigenvectors of the graph effectively evaluate φ. The dual-rail encoding of $\varphi(x) = f_{P(\varphi)}(x)$ promised by Theorem 2.5 will suffice to give a phase-estimation-based quantum algorithm for evaluating φ. The goal of this section is to make Theorem 2.5 more quantitative, which will enable us to analyze the algorithm's running time.

In particular, we will lower-bound the achievable squared support on either a_O or b_O of a unit-normalized $\lambda = 0$ eigenvector. This will enable the algorithm to detect if $\varphi(x) = 1$ by starting a quantum walk at a_O; if $\varphi(x) = 1$, then $|a_O\rangle$ will have large overlap with the $\lambda = 0$ eigenvector.

We also study eigenvalue-λ eigenvectors of $G_{P(\varphi)}(x)$, for $|\lambda| \neq 0$ sufficiently small. At small enough eigenvalues, the dual-rail encoding property of Theorem 2.5 still holds, in a different fashion. Note that since the graph is bipartite, we may take $\lambda > 0$ without loss of generality. For small enough $\lambda > 0$, it will turn out that the function evaluation corresponds to the output *ratio* $r_O \equiv a_O/b_O$. If $f_P(x) = 1$, then r_O is large and negative, roughly of order $-1/\lambda$. If $f_P(x) = 0$, then r_O is small and positive, roughly of order λ. Ultimately, the point of this analysis is to show that if the formula evaluates to false, then there do not exist any eigenvalue-λ eigenvectors supported on a_O for small enough $|\lambda| \neq 0$. This spectral gap will prevent the algorithm from outputting false positives.

Consider a span program P. Let us generalize the setting of Theorem 2.5 to allow P's inputs to be themselves span programs, as in Definition 2.9. Assume that for some x, every $\lambda \in [0, \Lambda)$ and each input $i \in I$, we have constructed unit-normalized states $|\psi_i(\lambda)\rangle$ satisfying the eigenvalue-λ constraints for all the ith subgraph's vertices except a_i.

Definition 3.1 (Subformula complexity)**.** *At $\lambda = 0$, for each input $i \in I_j$, let σ_i denote $|\psi_i\rangle$'s squared support on either a_j or b_i, depending on whether the input evaluates to true or false, respectively.*

For $\lambda > 0$, assume that the coefficients of $|\psi_i\rangle$ along the ith input edge are nonzero, and let $r_i = a_j/b_i$ be their ratio. If the literal associated to input i evaluates to false, then let $s_i = r_i/\lambda$; if it is true, then let $s_i = -1/(r_i\lambda)$.

For an input $i \in I$, its subformula complexity *is*

$$z_i = \max_x \max \left\{ 1/\sigma_i, \sup_{\lambda \in (0,\Lambda)} s_i \right\} . \tag{3.1}$$

For example, if z_i is small, then $|\psi_i(0)\rangle$ has large support on either a_i or b_i. In general, $z_i \geq 1$. If input i corresponds to a literal and not the output edge of another span program, then $z_i = 1$.

We construct a normalized state $|\psi_O(\lambda)\rangle$ that satisfies all the eigenvalue-λ eigenvector constraints of the composed graph, except Eq. (2.1d) at a_O. We construct $|\psi_O\rangle$ by putting together the scaled $|\psi_i\rangle$s and also

assigning coefficients to the vertices in G_P. Similarly to Eq. (3.1), define

$$z_O(x) = \max\left\{1/\sigma_O, \sup_{\lambda\in(0,\Lambda)} s_O\right\} \ , \tag{3.2}$$

where σ_O is the squared support of $|\psi_O(0)\rangle$ on a_O or b_O, and, for $\lambda > 0$, s_O is $-1/(r_O\lambda)$ or r_O/λ if $f(x)$ is true or false, respectively. We will relate $z_O = \max_x z_O(x)$ to the input complexities z_I (Theorem 3.7).

First of all, notice that if $|I_j| > 1$, then several of the input subgraphs share the vertex a_j. They must be scaled so that their coefficients at a_j all match, motivating the following definition.

Definition 3.2. *The* grouped input complexity *of $j \in J$ on input x is*

$$\tilde{z}_j(x) = \begin{cases} \max\left\{\sum_{i\in I_j} z_i, 1\right\} & \text{if } j \text{ evaluates to true} \\ \left(\sum_{false\ i\ \in\ I_j} z_i^{-1}\right)^{-1} & \text{otherwise} \end{cases} \tag{3.3}$$

Recall that grouped input j evaluates to true iff all inputs in I_j are true. (In the first case, we take the maximum with 1 to handle the case $I_j = \emptyset$.)

When j is false, some input $i \in I_j$ is false, so the coefficient at a_j must be set to zero at $\lambda = 0$. However, for each false $i \in I_j$, $|\psi_i\rangle$ can be scaled arbitrarily. The definition in Eq. (3.3) comes from choosing scale factors f_i in order to maximize the sum of the scaled coefficients on the vertices b_i, under the constraint that the total norm be one, $\sum_{i\in I_j} |f_i|^2 = 1$.

A few more definitions are needed to state Theorem 3.7.

Definition 3.3 (Asymptotic notation)**.** *Let $a \lesssim b$ mean that there exist constants c_1, c_2 such that $a \leq c_1 + b(1 + c_2|\lambda| \max_i z_i)$.*

Definition 3.4 (Matrix notations)**.** *For a given input x, let $\Pi(x) = \sum_{true\ j} |j\rangle\langle j|$ a projection onto the true grouped inputs, $\overline{\Pi}(x) = 1 - \Pi(x)$, and $\tilde{z}(x) = \sum_j \tilde{z}_j(x)|j\rangle\langle j|$ a diagonal matrix of the grouped input complexities. To simplify equations, we will generally leave implicit the dependence on x, writing Π, $\overline{\Pi}$ and $\tilde{z}$. Let $A = \binom{A_{OJ}}{A_{CJ}} = \sum_j |v_j\rangle\langle j|$ with columns the vectors $|v_j\rangle$.*

Definition 3.5 (Moore-Penrose pseudoinverse)**.** *For a matrix M, let M^+ denote its Moore-Penrose pseudoinverse. If the singular-value decomposition of M is $M = \sum_k m_k|k\rangle\langle k'|$ with all $m_k > 0$ and for sets of orthonormal vectors $\{|k\rangle\}$ and $\{|k'\rangle\}$, then $M^+ = \sum_k m_k^{-1}|k'\rangle\langle k|$. Note that $MM^+ = \sum_k |k\rangle\langle k|$ is the projection onto M's range.*

Definition 3.6 (Span program witness size)**.** *For span program P and input subformula complexities z_I, the witness size of P is* $\mathrm{wsize}(P) = \max_x \mathrm{wsize}(P, x)$*, where for an input x,* $\mathrm{wsize}(P, x)$ *is defined as follows:*

- *If $f_P(x) = 1$, then $|t\rangle \in \mathrm{Range}(A\Pi)$, so there is a witness $|w\rangle \in \mathbf{C}^{|J|}$ satisfying $A\Pi|w\rangle = |t\rangle$. Then* $\mathrm{wsize}(P, x)$ *is the minimum squared length, weighted by $\tilde{z}(x)^{1/2}$, of any such witness:*

$$\begin{aligned} \mathrm{wsize}(P, x) &= \min_{|w\rangle : A\Pi|w\rangle = |t\rangle} \|\tilde{z}^{1/2}|w\rangle\|^2 \\ &= \|(A\Pi\tilde{z}^{-1/2})^+|t\rangle\|^2 \ . \end{aligned} \tag{3.4}$$

- *If $f_P(x) = 0$, then $|t\rangle \notin \mathrm{Range}(A\Pi)$. Therefore there is a witness $|w'\rangle \in \mathbf{C}^{|C|+1}$ satisfying $\langle t|w'\rangle = 1$ and $\Pi A^\dagger|w'\rangle = 0$. Then*

$$\begin{aligned} \mathrm{wsize}(P, x) &= \min_{\substack{|w'\rangle : \langle t|w'\rangle = 1 \\ \Pi A^\dagger|w'\rangle = 0}} \|\tilde{z}^{1/2}A^\dagger|w'\rangle\|^2 \\ &= \|\big(1 + (\overline{\Pi}(A\tilde{z}^{1/2})^+A\tilde{z}^{1/2} - 1)^+\Pi\big)(A\tilde{z}^{1/2})^+|t\rangle\|^{-2} \ , \end{aligned} \tag{3.5}$$

  the inverse squared length of the projection of $(A\tilde{z}^{1/2})^+|t\rangle$ onto the intersection of $\overline{\Pi}$ and $\mathrm{Range}(\tilde{z}^{1/2}A^\dagger)$.

By $|w_x\rangle$, resp. $|w'_x\rangle$, we denote a witness for input x achieving the minimum in Eq. (3.4), resp. (3.5).

The span program witness size is easily computed on any given input x. Lemma A.3 below will give two alternative expressions for $\mathrm{wsize}(P,x)$. Now our main result is:

Theorem 3.7. *Consider a constant span program P. Assume that $\Lambda z_i \leq \epsilon$ for a small enough constant $\epsilon > 0$ to be determined and for all $i \in I$. Then*

$$z_O(x) \lesssim \mathrm{wsize}(P,x) \ . \tag{3.6}$$

For $\lambda = 0$, Eq. (3.6) says that the achievable squared magnitude on a_O or b_O of a normalized eigenvalue-zero eigenvector is at least $1/\mathrm{wsize}(P,x)$, up to small controlled terms. For $\lambda > 0$, Eq. (3.6) says that the ratio $r_O = a_O/b_O$ is either in $(0, \mathrm{wsize}(P,x)\lambda]$ or $(-\infty, -1/(\mathrm{wsize}(P,x)\lambda)]$, up to small controlled terms, depending on whether $f_P(x)$ is false or true.

Proof sketch of Theorem 3.7. At $\lambda = 0$, the proof of Theorem 3.7 is the same as that of Theorem 2.5, except scaling the inputs so as to maximize the squared magnitude on a_O or b_O. This maximization problem is essentially the same as the problems stated in Definition 3.6 (up to additive constants). The explicit expressions for the solutions follow by geometry.

For $\lambda > 0$, we solve the eigenvalue equations (2.1a,b,e) by inverting a matrix and applying the Woodbury formula. We argue that all inverses exist in the given range of λ. We obtain

$$r_O = a_O/b_O = \lambda + \langle o|\tilde{r}|o\rangle + \lambda\langle o|\tilde{r}{A_{CJ}}^\dagger X^{-1} A_{CJ}\tilde{r}|o\rangle \ ,$$

where $|o\rangle = {A_{OJ}}^\dagger$, $\tilde{r} = -\frac{1}{\lambda}\tilde{s}^{-1}\Pi + \lambda\tilde{s}\overline{\Pi}$ (with $\tilde{s}$ defined from s similarly to how $\tilde{z}$ is defined from z), and $X = A_{CJ}\tilde{s}^{-1}\Pi{A_{CJ}}^\dagger - \lambda^2 A_{CJ}\tilde{s}\overline{\Pi}{A_{CJ}}^\dagger - \lambda^2$. The largest term in X, $A_{CJ}\tilde{s}^{-1}\Pi{A_{CJ}}^\dagger$, is only invertible restricted to its range, $\Delta = A_{CJ}\Pi(A_{CJ}\Pi)^+$. Therefore, we compute the Taylor series in λ of the pseudoinverse of $\Delta X \Delta$ and of its Schur complement, $\big(X/(\Delta X\Delta)\big)$, separately, and then recombine them. The lowest-order term in the solution again corresponds to Definition 3.6 (if $f_P(x)$ is false, the $1/\lambda$ term is zero), and we bound the higher-order terms. □

The full proof of Theorem 3.7 is given in Appendix A.

Remark 3.8. *In case $f_P(x) = 0$, $A^\dagger|w'\rangle$ appears also in the "canonical form" of the span program P [KW93].*

The above analysis of span programs does not apply to the NOT gate, because the ability to complement inputs was assumed in Definition 2.1. Implementing the NOT gate $x \mapsto \overline{x}$ with a span program on the literal $\overline{x}$ would be circular. Therefore we provide a separate analysis.

Lemma 3.9 (NOT gate)**.** *Consider a NOT gate, implemented as two weight-one edges connected as in Definition 2.7. Assume $|\lambda| \leq 1/(\sqrt{2z_i})$. Then $z_O \lesssim z_i$.*

Proof. Analysis at $\lambda = 0$. If the input is true, then σ_i measures the squared support on a_i of a normalized $\lambda = 0$ eigenvector. Then $\sigma_O = \sigma_i$, since $a_i = b_O$ the output vertex. If the input is false, so $b_i = \sqrt{\sigma_i}$, then $b_i + a_O = 0$. Therefore, we simply need to renormalize: $\sigma_O = \sigma_i/(1+\sigma_i)$, or equivalently $\frac{1}{\sigma_O} = \frac{1}{\sigma_i} + 1$.

Analysis for small $\lambda > 0$. We are given $r_i = a_i/b_i$. The eigenvector equation is $b_i + a_O = \lambda a_i = \lambda b_O$. Therefore, $r_O = a_O/b_O = \lambda - 1/r_i$. If the input is false, so $s_i = r_i/\lambda$, then $s_O = -1/(\lambda r_O) = s_i/(1-\lambda^2 s_i)$. Therefore, $s_i < s_O \leq s_i(1 + 2\lambda^2 s_i)$ since $\lambda^2 s_i \leq 1/2$. If the input is true, so $s_i = -1/(\lambda r_i)$, then $s_O = r_O/\lambda = s_i + 1$.

Therefore $z_O \lesssim z_i$ as claimed. Note that w.l.o.g. we may assume there are never two NOT gates in a row in the formula φ, so the additive constants lost do not accumulate. □

4 Formula evaluation algorithm

In Section 4.1, we specify the gate set $\mathcal{S}$ (Definition 4.1) and define the correct notion of "balance" for a formula that includes different kinds of gates (Definition 4.5). These two definitions allow us to formulate the general statement of our results, Theorem 4.7, of which Theorem 1.1 is a corollary.

In Section 4.2, we present span programs of optimal witness size for each of the gates in $\mathcal{S}$. Theorem 4.16 in Section 4.3 plugs together the spectral analyses of the individual span programs to give a spectral analysis of $G(\varphi)$. Finally, we sketch in Section 4.4 how this implies a quantum algorithm, therefore proving Theorem 4.7.

4.1 General formula evaluation result

Definition 4.1 (Extended gate set $\mathcal{S}$)**.** *Let*

$$\mathcal{S}' = \left\{ \begin{array}{c} \textit{arbitrary 1-, 2-, or 3-bit gates,} \\ O(1)\textit{-fan-in EQUAL gates} \end{array} \right\}$$
$$\mathcal{S} = \left\{ \begin{array}{c} O(1)\textit{-size } \{\textit{AND, OR, NOT, PARITY}\} \\ \textit{read-once formulas composed onto} \\ \textit{the gates from } \mathcal{S}' \end{array} \right\} \tag{4.1}$$

Example 4.2. *The gate set $\mathcal{S}$ includes simple gates like AND, as well as substantially more complicated gates like* $\mathrm{MAJ}_3(x_1, x_2, x_3) \wedge (x_4 \oplus x_5 \oplus \cdots \oplus x_{k-1} \oplus (x_k \vee x_{k+1}))$*, provided $k = O(1)$. It does not include gates from $\mathcal{S}'$ composed onto gates from $\mathcal{S}$: for example* $\mathrm{MAJ}_3(x_1, x_2 \oplus x_3, x_4 \wedge x_5) \notin \mathcal{S}$.

To define "adversary-balanced" formulas, we need to define the nonnegative-weight quantum adversary bound.

Definition 4.3 (Nonnegative adversary bound)**.** *Let $f : \{0,1\}^k \to \{0,1\}$. Define*

$$\mathrm{Adv}(f) = \max_{\substack{\Gamma \geq 0 \\ \Gamma \neq 0}} \frac{\|\Gamma\|}{\max_i \|\Gamma \circ D_i\|} \ , \tag{4.2}$$

where $\Gamma \circ D_i$ denotes the entrywise matrix product between Γ and D_i a zero-one-valued matrix defined by $\langle x|D_i|y\rangle = 1$ if and only if bitstrings x and y differ in the ith coordinate, for $i \in \{1, \ldots, k\}$. The maximum is over all $2^k \times 2^k$ symmetric matrices Γ with nonnegative entries satisfying $\langle x|\Gamma|y\rangle = 0$ if $f(x) = f(y)$.

The motivation for this definition is that $\mathrm{Adv}(f)$ gives a lower bound on the number of queries to the *phase-flip input oracle*

$$O_x : |b, i\rangle \mapsto (-1)^{b \cdot x_i} |i\rangle \tag{4.3}$$

required to evaluate f on input x.

Theorem 4.4 (Adversary lower bound [Amb06a, BSS03])**.** *The two-sided ϵ-bounded error quantum query complexity of function f, $Q_\epsilon(f)$, is at least* $\frac{1-2\sqrt{\epsilon(1-\epsilon)}}{2}\mathrm{Adv}(f)$.

To match the lower bound of Theorem 4.4, our goal will be to use $O(\mathrm{Adv}(\varphi))$ queries to evaluate φ.

Definition 4.5 (Adversary-balanced formula)**.** *For a gate g in formula φ, let φ_g denote the subformula of φ rooted at g. Define φ to be adversary-balanced if for every gate g, the adversary lower bounds for its input subformulas are the same; if g has children $h_1, \ldots, h_k$, then* $\mathrm{Adv}(\varphi_{h_1}) = \cdots = \mathrm{Adv}(\varphi_{h_k})$.

Definition 4.5 is motivated by a version of an adversary composition result [Amb06a, HLŠ07]:

Theorem 4.6 (Adversary composition [HLŠ07])**.** *Let $f = g \circ (h_1, \ldots, h_k)$, where* $\mathrm{Adv}(h_1) = \cdots = \mathrm{Adv}(h_k)$ *and $\circ$ denotes function composition. Then* $\mathrm{Adv}(f) = \mathrm{Adv}(g)\mathrm{Adv}(h_1)$.

Gate f	$\mathrm{Adv}(f)$	Gate f	$\mathrm{Adv}(f)$
0	0	$\mathrm{MAJ}_3(x_{[3]}) = (x_1 \wedge x_2) \vee ((x_1 \vee x_2) \wedge x_3)$	2
x_1	1	$\mathrm{EQUAL}_3(x_{[3]}) = (x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3})$	$3/\sqrt{2}$
$x_1 \wedge x_2$	$\sqrt{2}$	$(x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2})$	$\sqrt{3+\sqrt{3}}$
$x_1 \oplus x_2$	2	$x_1 \vee (x_2 \oplus x_3)$	$\sqrt{5}$
$x_1 \wedge x_2 \wedge x_3$	$\sqrt{3}$	$x_1 \oplus (x_2 \wedge x_3)$	$1+\sqrt{2}$
$x_1 \vee (x_2 \wedge x_3)$	$\sqrt{3}$	$\mathrm{EXACT}_{2 \text{ of } 3}(x_{[3]}) = \mathrm{MAJ}_3(x_{[3]}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$	$\sqrt{7}$
$(x_3 \wedge x_2) \vee (\overline{x_3} \wedge x_1)$	2	$x_1 \oplus x_2 \oplus x_3$	3

Figure 5: Binary gates on up to three bits. Up to equivalences—permutation of inputs, complementation of some or all inputs or output—there are fourteen binary gates on three inputs x_1, x_2, x_3. Adversary bounds $\mathrm{Adv}(f)$ for all functions f on up to four bits have been computed by [HLŠ06], and see [RŠ07].

If φ is adversary-balanced, then by Theorem 4.6 $\mathrm{Adv}(\varphi_g)$ is the product of the gate adversary bounds along any non-self-intersecting path χ from g up to an input, $\mathrm{Adv}(\varphi_g) = \prod_{h \in \chi} \mathrm{Adv}(h)$. Note that $\mathrm{Adv}(\neg f) = \mathrm{Adv}(f)$, so NOT gates can be inserted anywhere in an adversary-balanced formula.

The main result of this paper is

Theorem 4.7 (Main result)**.** *There exists a quantum algorithm that evaluates an adversary-balanced formula $\varphi(x)$ over $\mathcal{S}$ using $O(\mathrm{Adv}(\varphi))$ queries to the phase-flip input oracle O_x. After efficient classical preprocessing independent of the input x, and assuming $O(1)$-time coherent access to the preprocessed classical string, the running time of the algorithm is $\mathrm{Adv}(\varphi)(\log \mathrm{Adv}(\varphi))^{O(1)}$.*

From Figure 5, the adversary bound $\mathrm{Adv}(\mathrm{MAJ}_3) = 2$. By Theorem 4.6 the adversary bound for the balanced MAJ_3 formula of depth d is 2^d. Theorem 1.1 is therefore essentially a corollary of Theorem 4.7 (for the balanced MAJ_3 formula, coherent access to a preprocessed classical string is not needed).

4.2 Optimal span programs for gates in $\mathcal{S}$

In this section, we will substitute specific span programs into Definition 3.6, in order to prove:

Theorem 4.8. *Let $\mathcal{S}$ be the gate set of Definition 4.1. For every gate $f \in \mathcal{S}$, there exists a span program P computing $f_P = f$, such that the witness size of P (Definition 3.6) on equal input complexities $z_i = 1$ is*

$$\mathrm{wsize}(P) = \mathrm{Adv}(f) \ . \tag{4.4}$$

$\mathrm{Adv}(f)$ is the adversary bound for f (Definition 4.3).

Proof. We analyze five of the fourteen inequivalent binary functions on at most three bits, listed in Figure 5: 0 and x_1 (both trivial), the MAJ_3 gate (Claim 4.9), the k-bit EQUAL_k gate (Claim 4.10), and a certain three-bit function, $g(x) = (x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2})$ (Claim 4.11).

Claim 4.9. *Let P_{MAJ_3} be the span program from Example 2.2. Then* $\mathrm{wsize}(P_{\mathrm{MAJ}_3}) = 2 = \mathrm{Adv}(\mathrm{MAJ}_3)$.

Proof. Substitute P_{MAJ_3} into Definition 3.6. Some of the witness vectors are $|w'_{000}\rangle = (1,0)$, $|w'_{100}\rangle = (1, -1/\sqrt{3})$, and $|w_{110}\rangle = (e^{-i\pi/6}, e^{i\pi/6}, 0)$, $|w_{111}\rangle = (1,1,1)/\sqrt{3}$. □

Claim 4.10. *Letting $\alpha = \sqrt[4]{k-1}$, the span program*

$$\begin{aligned} & X_J = (\{x_1, x_2, \ldots, x_k\} \ \{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_k}\}) \\ t = (1)\,, \quad & v_J = (\qquad \alpha \qquad\qquad \alpha \qquad) \end{aligned}$$

computes EQUAL_k with witness size $\frac{k}{\sqrt{k-1}} = \mathrm{Adv}(\mathrm{EQUAL}_k)$.[1]

[1] The optimal adversary matrix Γ comes from the $2 \times 2k$ matrix $\left(\begin{smallmatrix} 1 & 1 & \cdots & a & a & \cdots \\ a & a & \cdots & 1 & 1 & \cdots \end{smallmatrix}\right)$, where the rows correspond to inputs 0^k and 1^k, and the columns correspond to inputs of Hamming weight 1 then $k-1$, and $a = 1/(k-1)$.

Proof. Substitute into Definition 3.6. The witnesses are $|w_{0^k}\rangle = \left(0, \frac{\sqrt{k}}{\alpha}\right)$, $|w_{1^k}\rangle = \left(\frac{\sqrt{k}}{\alpha}, 0\right)$ and $|w'_x\rangle = (1)$ for $x \notin \{0^k, 1^k\}$. □

Claim 4.11. *Let* $g(x) = (x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2})$. *Letting* $\alpha_1 = \sqrt[4]{1 + \frac{1}{\sqrt{3}}}$ *and* $\alpha_2 = \sqrt{\sqrt{3} - 1}$, *the span program*

$$X_J = (\{x_1, x_2\}\ \{\overline{x_1}, \overline{x_2}\}\ \{x_3\})$$
$$t = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad v_J = \begin{pmatrix} \alpha_1 & \alpha_1\alpha_2 & 0 \\ \alpha_2 & 0 & 1 \end{pmatrix}$$

computes g *with witness size* $\sqrt{3 + \sqrt{3}} = \mathrm{Adv}(g)$.[2]

Proof. By substitution into Definition 3.6. □

For all the remaining gates in $\mathcal{S}$, it suffices to analyze the NOT gate (Lemma 3.9), and OR and PARITY gates on *unbalanced* inputs (Lemma 4.12). That is, we allow z_1 and z_2 to be different, with $z_1/z_2, z_2/z_1 = O(1)$. For functions b and b' on disjoint inputs, $\mathrm{Adv}(b \oplus b') = \mathrm{Adv}(b) + \mathrm{Adv}(b')$, and $\mathrm{Adv}(b \vee b') = \sqrt{\mathrm{Adv}(b)^2 + \mathrm{Adv}(b')^2}$ [BS04, HLŠ07]; we obtain matching upper bounds for span program witness size.

Lemma 4.12. *Consider* $f(x, x') = f'(b(x), b'(x'))$, *with* $f' \in \{\mathrm{PARITY}, \mathrm{OR}\}$, *and* b *and* b' *functions on* $O(1)$ *bits. Assume that there exist span programs* P_b *and* $P_{b'}$ *for* b *and* b' *with respective witness sizes* $B = \mathrm{wsize}(P_b)$ *and* $B' = \mathrm{wsize}(P_{b'})$. *Then there exists a span program* P *for* f *with witness size* $\mathrm{wsize}(P) = B + B'$ *if* $f' = \mathrm{PARITY}$, *or* $\sqrt{B^2 + B'^2}$ *if* $f' = \mathrm{OR}$.

Proof. Substitute the following span programs with zero constraints into Definition 3.6:

$$P_{\mathrm{PARITY}} : t = (1), \quad \begin{array}{c} X_J = (\ \{x_1, \overline{x_2}\}\ \ \{\overline{x_1}, x_2\}\ ) \\ v_J = (\ \ 1 \qquad 1\ \ ) \end{array},$$

$$P_{\mathrm{OR}} : t = (1), \quad \begin{array}{c} X_J = (\ \ \{x_1\} \qquad \{x_2\}\ \ ) \\ v_J = \left(\frac{\sqrt{B}}{\sqrt[4]{B^2+B'^2}}\ \ \frac{\sqrt{B'}}{\sqrt[4]{B^2+B'^2}}\right) \end{array}.$$

The witness vectors for PARITY are $|w'_{00}\rangle = (1)$ and $|w_{10}\rangle = (\sqrt{B^2 + B'^2}, 0)$, and the witness vectors for OR are $|w'_{00}\rangle = (1)$, $|w_{10}\rangle = (\sqrt[4]{B^2 + B'^2}, 0)$, and $|w_{11}\rangle = (1, 1) \cdot \frac{1}{2}\sqrt[4]{B^2 + B'^2}$. □

Then, e.g., the function $\mathrm{EXACT}_{2\text{ of }3}(x_1, x_2, x_3) = \mathrm{MAJ}_3(x_1, x_2, x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$, so Lemma 4.12 implies a span program for $\mathrm{EXACT}_{2\text{ of }3}$ with witness size $\sqrt{7} = \sqrt{\mathrm{wsize}(\mathrm{MAJ}_3)^2 + \mathrm{wsize}(\mathrm{OR}_3)^2}$. □

Remark 4.13. *Our procedure for analyzing a function* f *has been as follows:*

1. *First determine a span program* P *computing* $f_P = f$. *The simplest span program is derived from the minimum-size* $\{AND, OR, NOT\}$ *formula for* f.
2. *Next, compute* $\mathrm{wsize}(P, x)$ *for each input* x, *as a function of the variable weights of* P.
3. *Finally, optimize the free weights of* P *to minimize* $\mathrm{wsize}(P) = \max_x \mathrm{wsize}(P, x)$. *For example, note that scaling* A_{OJ} *up helps the true cases in Definition 3.6, and hurts the false cases; therefore choose a scale to balance the worst true case against the worst false case.*

   We respect the symmetries of f *during optimization. On the other hand, if two literals are not treated symmetrically by* f, *then we do not group them together in any grouped input* X_j. *For example, in Claim 4.11 we do not group* x_3 *together with* x_1 *and* x_2 *in* X_1.

[2] The optimal adversary matrix Γ comes from the matrix $\begin{pmatrix} 1 & a & \sqrt{3} \\ 1 & a & \sqrt{3} \\ b & 0 & \sqrt{3} \end{pmatrix}$, where $a = \left(\frac{1}{2}(5 - \sqrt{13 - 6\sqrt{3}})\right)^{1/2}$, $b = \frac{1}{2}\left(-1 - \sqrt{3} + \sqrt{2(8 + \sqrt{3})}\right)$, and the rows correspond to inputs 011, 101, 110, the columns to inputs 000, 001, 111.

Remark 4.14. *The proof of Theorem 4.8 uses separate analyses for* EQUAL_k, MAJ_3 *and* g *because the upper bounds from Lemma 4.12 for these functions do not match the adversary lower bounds. For example, from Figure 5 the smallest* $\{AND, OR, NOT\}$ *formula for* MAJ_3 *has five inputs,* $\mathrm{MAJ}_3(x_{[3]}) = (x_1 \wedge x_2) \vee ((x_1 \vee x_2) \wedge x_3)$. *Lemma 4.12 therefore gives a span program* P *for* MAJ_3 *with witness size* $\mathrm{wsize}(P) = \sqrt{5}$. *In fact, optimizing the weights of this* P *gives a span program with witness size* $\sqrt{3+\sqrt{2}} < \sqrt{5}$*; the worst-case inputs of the read-once formula* $(x_1 \wedge x_2) \vee ((x_4 \vee x_5) \wedge x_3)$ *do not arise under the promise that* $x_4 = x_1$ *and* $x_5 = x_2$. *However, this is still worse than the span program* P_{MAJ_3} *of Example 2.2, with* $\mathrm{wsize}(P_{\mathrm{MAJ}_3}) = 2$.

Remark 4.15. *Lemma 4.12 implies that any* $\{AND, OR, NOT\}$ *formula of bounded size has a span program with witness size the square root of the sum of the squares of the input complexities. We conjecture that this holds even for formulas with size* $\omega(1)$*; see [Amb06b, ACR+07] for special cases.*

4.3 Span program spectral analysis of φ

Theorem 4.16. *Consider an adversary-balanced formula* φ *on the gate set* $\mathcal{S}$*, with adversary bound* $\mathrm{Adv}(\varphi)$. *Let* P *be the composed span program computing* $f_P = \varphi$. *For an input* $x \in \{0,1\}^N$*, recall the definition of the weighted graph* $G_P(x)$ *from Theorem 2.5; if the literal on an input edge evaluates to true, then delete that edge from* G_P. *Let* $\tilde{G}_P(x)$ *be the same as* $G_P(x)$ *except with the weight on the output edge* (a_O, b_O) *set to* $w = \epsilon_w / \sqrt{\mathrm{Adv}(\varphi)}$ *(instead of weight one), where* $\epsilon_w > 0$ *is a sufficiently small constant. Then,*

- *If* $\varphi(x) = 1$*, there exists a normalized eigenvalue-zero eigenvector of the adjacency matrix* $A_{\tilde{G}_P(x)}$ *with* $\Omega(1)$ *support on the output vertex* a_O.
- *If* $\varphi(x) = 0$*, then for some small enough constant* $\epsilon > 0$, $A_{\tilde{G}_P(x)}$ *does not have any eigenvalue-*λ *eigenvectors supported on* a_O *or* b_O *for* $|\lambda| \leq \epsilon / \mathrm{Adv}(\varphi)$.

Proof. The proof of Theorem 4.16 has two parts. First, we will prove by induction that $z_g = O(\mathrm{Adv}(\varphi_g))$. Then, by considering the last eigenvector constraint, $\lambda a_O = w b_O$, we either construct the desired eigenvector or derive a contradiction, depending on whether $\varphi(x)$ is true or false.

Base case. Consider an input x_i to the formula φ. If $x_i = 1$, then the corresponding input edge (a_j, b_i) is not in $G_P(x)$. In particular, the input i does not contribute to the expression for $\tilde{z}_j(x)$ in Eq. (3.3), so z_i may be left undefined. If $x_i = 0$, then the input edge (a_j, b_i) is in $G_P(x)$. The eigenvalue-λ equation at b_i is $\lambda b_i = a_j$. For $\lambda = 0$, this is just $a_j = 0$, so let $\sigma_i = 1$. For $\lambda > 0$, this is $r_i = \lambda s_i = a_j / b_i = \lambda$, so $s_i = z_i = 1$.

Induction. Assume that $|\lambda| \leq \epsilon / \mathrm{Adv}(\varphi)$, for some small enough constant $\epsilon > 0$.

Consider a gate g. Let $h_1, \ldots, h_k$ be the inputs to g. Let φ_g denote the subformula of φ based at g. By Theorem 3.7 and Theorem 4.8, the output bound z_g satisfies

$$z_g \lesssim \mathrm{Adv}(g) \max_i z_{h_i} \ , \tag{4.5}$$

or equivalently

$$z_g \leq c_1 + \mathrm{Adv}(g)(\max_i z_{h_i})(1 + c_2 \cdot |\lambda| \mathrm{Adv}(\varphi_g)) \tag{4.6}$$

for certain constants c_1, c_2. Different kinds of gates give different constants in Eq. (4.6), but since the gate set is finite, all constants are uniformly $O(1)$.

Since $|\lambda| \leq \epsilon / \mathrm{Adv}(\varphi)$, the recurrence Eq. (4.6) has solution

$$z_g \leq O\left(\max_\chi \prod_{h \in \chi} \mathrm{Adv}(h) \Big(1 + \epsilon' \frac{\mathrm{Adv}(\varphi_h)}{\mathrm{Adv}(\varphi)} \Big) \right) \ ,$$

where the maximum is taken over the choice of χ a non-self-intersecting path from g up to an input. Because φ is by assumption adversary balanced (Definition 4.5), $\prod_{h \in \chi} \mathrm{Adv}(h) = \mathrm{Adv}(\varphi_g)$ (Theorem 4.6). Also, $\prod_{h \in \chi} (1 + \epsilon' \frac{\mathrm{Adv}(\varphi_h)}{\mathrm{Adv}(\varphi)}) = O(1)$. Therefore, the solution satisfies

$$z_g = O(\mathrm{Adv}(\varphi_g)) \ . \tag{4.7}$$

Final amplification step. Assume $\varphi(x) = 1$. Then by Eq. (4.7), there exists a normalized eigenvalue-zero eigenvector of the graph $G_P(x)$ with squared amplitude $|a_O|^2 \geq \sigma_O = 1/O(\mathrm{Adv}(\varphi))$. Recall that $w = \epsilon_w/\sqrt{\mathrm{Adv}(\varphi)}$ is the weight of the output edge (a_O, b_O) of P in $\tilde{G}_P(x)$, and let $\hat{a}_O = wa_O$. The $\lambda = 0$ eigenvector equations for $\tilde{G}_P(x)$ are the same as those for $G_P(x)$, except with $\hat{a}_O$ in place of a_O. Therefore, we may take $|\hat{a}_O|^2 = 1/O(\mathrm{Adv}(\varphi))$, so for a normalized eigenvalue-zero eigenvector of $\tilde{G}_P(x)$, $|a_O|^2 = \Omega(1)$. By reducing the weight of the output edge from 1 to w, we have amplified the support on a_O up to a constant.

Now assume that $\varphi(x) = 0$. By Theorem 2.5, there does not exist any eigenvalue-zero eigenvector supported on a_O. Also $b_O = 0$ at $\lambda = 0$ by the constraint $\lambda a_O = wb_O$. For $\lambda \neq 0$, $|\lambda| \leq \epsilon/\mathrm{Adv}(\varphi)$, Eq. (4.7) implies that in any eigenvalue-λ eigenvector for $\tilde{G}_P(x)$, either $\hat{a}_O = b_O = 0$ or the ratio $|\hat{a}_O/b_O| \leq |\lambda| \cdot O(\mathrm{Adv}(\varphi))$, so

$$|a_O/b_O| \leq c_3 \cdot \frac{|\lambda|}{w}\mathrm{Adv}(\varphi) \tag{4.8}$$

for some constant c_3 that does not depend on w. We have not yet used the eigenvector equation at a_O, $\lambda a_O = wb_O$. Combining this equation with Eq. (4.8), we get $w^2 \leq c_3\lambda^2\mathrm{Adv}(\varphi) \leq c_3\epsilon^2/\mathrm{Adv}(\varphi)$. Substituting $w = \epsilon_w/\sqrt{\mathrm{Adv}(\varphi)}$, this is a contradiction provided we set ϵ_w so $\epsilon_w^2 > c_3\epsilon^2$. Therefore, the adjacency matrix of $\tilde{G}_P(x)$ cannot have an eigenvalue-λ eigenvector supported on a_O or b_O. □

4.4 Quantum algorithm

We apply Theorem 4.16 and the Szegedy correspondence between discrete- and continuous-time quantum walks [Sze04] to design the optimal quantum algorithm needed to prove Theorem 4.7. The approach is similar to that used for the NAND formula evaluation algorithm of [CRŠZ07], with only technical differences. Full details are given in Appendix B.

The main idea is to construct a discrete-time quantum walk $U_x = \tilde{O}_x U_{0^N}$ on the directed edges of G_P whose spectrum and eigenvectors correspond exactly to those of $A_{\tilde{G}_P(x)}$. Here U_{0^N} is a fixed unitary operator only depending on the formula graph $A_{\tilde{G}_P(0^N)}$, which can be implemented efficiently without access to the input x, and $\tilde{O}_x$ is defined by

$$\tilde{O}_x|v,w\rangle = \begin{cases} (-1)^{x_{i(v)}}|v,w\rangle & \text{if } v \text{ is a leaf} \\ |v,w\rangle & \text{otherwise} \end{cases} \tag{4.9}$$

where $i(v)$ is the index of the input variable corresponding to the leaf v. One call to $\tilde{O}_x$ can be implemented using one call to the standard phase-flip oracle O_x of Eq. (4.3).

Now starting at the output edge $|a_O, b_O\rangle$, run phase estimation [CEMM98] on U_x with precision $\delta_p = O(1/\mathrm{Adv}(\varphi))$ and error δ_e a small enough constant. Output "$\varphi(x) = 1$" iff the output phase is zero. The query complexity of this algorithm is $O(1/\delta_p) = O(\mathrm{Adv}(\varphi))$. The first part of Theorem 4.16 implies completeness, because the initial state has constant overlap with an eigenstate of U_x with phase zero. The second part of Theorem 4.16 implies soundness, because the spectral gap away from zero is greater than the precision δ_p.

5 Extensions and open problems

Theorem 4.7 can be extended in several directions, and there are many open problems including those from [CRŠZ07]. For example, is the eigenvalue-zero eigenstate useful for extracting witness information? We would like to raise several other questions.

5.1 Four-bit gates

The gate set $\mathcal{S}$ includes all three-bit binary gates. What about four-bit gates? Up to symmetries, there are 208 inequivalent binary functions that depend on exactly four input bits $x_1, \ldots, x_4$. The functions we have considered so far are listed at the webpage [RŠ07]. To summarize,

- Thirty of the functions can be written as a PARITY or OR of two subformulas on disjoint inputs. These functions are already included in the gate set $\mathcal{S}$ (Definition 4.1).
- For 25 additional functions, we have found a span program with witness size matching the adversary lower bound. These functions can be added to $\mathcal{S}$ without breaking Theorem 4.7.
- For 20 of the remaining functions, we have found a span program with complexity beating the square-root of the minimum {AND, OR, NOT} formula size, but not matching the adversary lower bound.

Example 5.1 (Threshold 2 of 4)**.** *In analogy to Example 2.2, one might consider the span program*

$$X_J = (\{x_1\}\ \{x_2\}\ \{x_3\}\ \{x_4\})$$
$$t = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad v_J = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \end{pmatrix} .$$

*This span program computes Threshold$_{2\ of\ 4}(x_{[4]})$—*MAJ$_3$ *is Threshold$_{2\ of\ 3}$—but it is not optimal. Intuitively, the problem is that the different pairs of inputs are not symmetrical. An optimal span program, with witness size $\sqrt{6}$, is*

$$X_J = (\{x_1\}\ \{x_1\}\quad \{x_2\}\ \{x_2\}\quad \{x_3\}\ \{x_3\}\quad \{x_4\}\ \{x_4\})$$
$$t = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad v_J = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & i & -i & i & i \\ i & -i & i & i & 1 & 1 & 1 & -1 \end{pmatrix}$$

It was derived by embedding a four-simplex symmetrically in the 2×2 unitary matrices, in correct analogy to Example 2.2. This embedding gives a span program over an extension ring of $\mathbf{C}$ *that, following [KW93, Theorem 12] and [BGW99, Prop. 2.8], can be simulated by a span program over the base ring.*

The Hamming-weight threshold functions Threshold$_{h\ of\ k} : \{0,1\}^k \to \{0,1\}$ defined by

$$\text{Threshold}_{h\ of\ k}(x) = \begin{cases} 1 & \text{if } |x| \geq h \\ 0 & \text{if } |x| < h \end{cases}$$

are functions that we currently have an understanding of only for $h \in \{0, 1, k\}$ and a partial understanding of for $h \in \{2, k-1\}$. Another function of particular interest is the six-bit Kushilevitz function [HLŠ07, Amb06a]. It seems that k-bit gates are inevitably going to require more involved techniques to evaluate optimally, for k large enough. It may well be that four-bit gates are already interesting in this sense.

5.2 Unbalanced formulas

Can the restriction that the gates have adversary-balanced inputs be significantly weakened? So far, we have only analyzed the PARITY and OR gates for unbalanced inputs, in Lemma 4.12. For the MAJ$_3$ gate, we have found an optimal span program for the case in which only two of the inputs are balanced:

Lemma 5.2. *Let $f(x, x', x'') =$* MAJ$_3(b(x), b'(x'), b''(x''))$ *with b, b', b'' functions on $O(1)$ bits computed by span programs $P_b, P_{b'}, P_{b''}$ with witness sizes $B =$* wsize$(P_b) =$ Adv$(b) =$ wsize$(P_{b'}) =$ Adv(b') *and $B'' =$* wsize$(P_{b''}) =$ Adv(b'')*. Let $\beta = B''/B$ and $\alpha = \frac{1}{2\sqrt{2}}(\sqrt{8+\beta^2} - \beta)^{1/2}$. Then there exists a span program P for f with* wsize$(P) = \frac{1}{2}\big(\sqrt{8+\beta^2} + \beta\big)B =$ Adv(f)*:*

$$X_J = (\{x_1\}\ \{x_2\}\quad \{x_3\}\quad)$$
$$t = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad v_J = \begin{pmatrix} \alpha & \alpha & \sqrt{\frac{1}{2} + \beta\alpha^2} \\ i & -i & 2\alpha \end{pmatrix} .$$

Therefore, for example, the four-bit gates $\mathrm{MAJ}_3(x_1, x_2, x_3 \wedge x_4)$ and $\mathrm{MAJ}_3(x_1, x_2, x_3 \oplus x_4)$ can be added into $\mathcal{S}$ without affecting the correctness of Theorem 4.7 (see Section 5.1). However, we do not have an understanding of MAJ_3 when all three input complexities differ. In this case, the formula for the adversary lower bound is substantially more complicated, and we do not have a matching span program.

For other gates, with the exception of PARITY and OR, we know similarly little. For a highly unbalanced formula with large depth, there is the further problem of whether the formula can be rebalanced without increasing its adversary lower bound too much [CRŠZ07].

5.3 Witness vectors and the adversary bound

The witnesses in Definition 3.6 have an interesting property related to a dual version of the adversary bound [LM04, ŠS06]: Assume that all $|X_j| = 1$ and $z = 1$. For $x, y \in \{0,1\}^n$ with $f_P(x) = 1$, $f_P(y) = 0$, consider the witnesses $|w_x\rangle$, $|w'_y\rangle$ achieving the minima in Eqs. (3.4), (3.5), and let $|w_y\rangle = A^\dagger |w'_y\rangle$. Then $|w_x\rangle = \Pi(x)|w_x\rangle$ and $\overline{\Pi}(y)|w_y\rangle = |w_y\rangle$, so

$$\langle w_x|\Pi(x)\overline{\Pi}(y)|w_y\rangle = \langle w_x|A^\dagger|w'_y\rangle = \langle t|w'_y\rangle = 1 \ .$$

Therefore, if we define $p_x(i) = \frac{1}{\||w_x\rangle\|^2} \sum_{\substack{j:\ X_j=\{x_i\} \\ \vee\ X_j=\{\overline{x_i}\}}} |\langle j|w_x\rangle|^2$ for each x (for both true and false $f_P(x)$) and for $i \in [n]$, then we get a feasible set of probability distributions for the minimax formulation of the adversary bound [ŠS06]. If $\mathrm{wsize}(P) = \mathrm{Adv}(f_P)$, then this set of probability distributions is optimal.

In this paper, we only use the adversary bound with nonnegative weights $\mathrm{Adv}(f)$. In fact, Hoyer, Lee and Špalek showed that Eq. (4.2) still provides a lower bound on the quantum query complexity even when one removes the restriction that the entries of Γ be nonnegative [HLŠ07]. This more general adversary bound, $\mathrm{Adv}^{\pm}(f)$ is clearly at least $\mathrm{Adv}(f)$. Theorem 4.6 is not known to hold for $\mathrm{Adv}^{\pm}$ composition; however, under the conditions of the theorem, it is known that $\mathrm{Adv}^{\pm}(f) \geq \mathrm{Adv}^{\pm}(g)\mathrm{Adv}^{\pm}(h_1)$. For every three-bit function f, no advantage is gained by allowing negative weights: $\mathrm{Adv}^{\pm}(f) = \mathrm{Adv}(f)$. For most functions f on four bits, though, $\mathrm{Adv}^{\pm}(f) > \mathrm{Adv}(f)$ [HLŠ06]. Therefore, one gets an asymptotically higher lower bound for formulas with such functions as gates than using Adv. However, for no function f with $\mathrm{Adv}(f) < \mathrm{Adv}^{\pm}(f)$ do we have a span program that matches $\mathrm{Adv}^{\pm}(f)$. The dual formulation of $\mathrm{Adv}^{\pm}$ cannot be expressed using probability distributions and one therefore cannot hope for a simple correspondence with the witnesses like described above.

Both variants of the adversary bound, Adv and $\mathrm{Adv}^{\pm}$, can be expressed as optimal solutions of certain semidefinite programs. Can one find a semidefinite formulation of span program witness size?

5.4 Eliminating the preprocessing

In many cases for φ, the preprocessing step of algorithm $\mathcal{ALG}$ can be eliminated. Because φ is an adversary-balanced formula on a known gateset, a decomposition through Theorem B.4 can be computed separately for each gate of $\mathcal{S}$ and then put together at runtime. This decomposition is *not* the decomposition of Claim B.2, which involves global properties of φ like $\|A'\|$. For an example, see the exactly balanced NAND tree algorithm in [CRŠZ07].

The decompositions can be combined because all the weights of gate input/output edges are one. This is quite different from the case of unbalanced NAND trees considered by [CRŠZ07], in which the weight of an input edge depends on the subformula entering it.

5.5 Arbitrary {AND, OR, NOT, PARITY} formulas

Some of the conditions on the gates in $\mathcal{S}$ (Definition 4.1) can be loosened. For example, $\mathcal{S}$ includes as single gates $O(1)$-size {AND, OR, NOT, PARITY} formulas on inputs that are themselves possibly elements of $\mathcal{S}'$. Let f be such a gate, $f = g \circ (h_1, \ldots, h_k)$ with g an {AND, OR, NOT, PARITY} formula of size $O(1)$, and each h_i either the identity or a gate from $\mathcal{S}'$. We have assumed that all the inputs to f have equal adversary

bounds. However, the stated proof works equally well if only each h_i has inputs with equal adversary bounds, provided the inputs to h_i and to $h_{i'}$ have adversary bounds that differ by at most a constant factor.

We believe that the assumption that g be of size $O(1)$ can also be significantly weakened. A stronger analysis like that of [CRŠZ07] for "approximately balanced" {AND, OR, NOT} formulas can presumably also be applied with PARITY gates. We have avoided this analysis to simplify the proofs, and to focus on the main novelty of this paper, the extended gate sets.

For {AND, OR, NOT, PARITY} formulas that are not "approximately balanced," rebalancing will typically be required. We have not investigated how the formula rebalancing procedures of [BCE91, BB94] affect the formula's adversary bound. In [CRŠZ07], it sufficed to consider the effect on the formula size, because the adversary bound for any {AND, OR, NOT} formula on N inputs is always $\sqrt{N}$.

5.6 New algorithms based on span programs

We have begun the development of a new framework for quantum algorithms based on span programs. In this paper, we have only composed bounded-size span programs evaluating functions each on $O(1)$ bits. An intriguing question is, do there exist interesting quantum algorithms based directly on asymptotically large span programs? Some candidate problems may be found in [BGW99, BGP96], although note that the quantum algorithm works for span programs over $\mathbf{C}$ that need not be monotone.

Acknowledgements

We thank Troy Lee for pointing out span programs to us. B.R. would like to thank Andrew Childs, Sean Hallgren, Cris Moore, David Yonge-Mallo and Shengyu Zhang for helpful conversations.

References

[ABO99] Eric Allender, Robert Beals, and Mitsunori Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Computational Complexity*, 8:99–126, 1999. Preliminary version in *Proc. 28th ACM STOC*, 1996.

[ACR+07] Andris Ambainis, Andrew M. Childs, Ben W. Reichardt, Robert Špalek, and Shengyu Zhang. Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. In *Proc. 48th IEEE FOCS*, pages 363–372, 2007.

[Amb06a] Andris Ambainis. Polynomial degree vs. quantum query complexity. *J. Comput. Syst. Sci.*, 72(2):220–238, 2006. Preliminary version in *Proc. 44th IEEE FOCS*, 2003.

[Amb06b] Andris Ambainis. Quantum search with variable times. arXiv:quant-ph/0609168, 2006.

[Amb07] Andris Ambainis. A nearly optimal discrete query quantum algorithm for evaluating NAND formulas. arXiv:0704.3628 [quant-ph], 2007.

[BB94] Maria Luisa Bonet and Samuel R. Buss. Size-depth tradeoffs for Boolean formulae. *Information Processing Letters*, 49(3):151–155, 1994.

[BCE91] Nader H. Bshouty, Richard Cleve, and Wayne Eberly. Size-depth tradeoffs for algebraic formulae. In *Proc. 32nd IEEE FOCS*, pages 334–341, 1991.

[BGP96] Amos Beimel, Anna Gál, and Mike Paterson. Lower bounds for monotone span programs. *Computational Complexity*, 6:29–45, 1996. Preliminary version in *Proc. 36th IEEE FOCS*, 1995.

[BGW99] László Babai, Anna Gál, and Avi Wigderson. Superpolynomial lower bounds for monotone span programs. *Combinatorica*, 19(3):301–319, 1999. Preliminary version in *Proc. 28th ACM STOC*, 1996.

[BS04] Howard Barnum and Michael Saks. A lower bound on the quantum query complexity of read-once functions. *J. Comput. Syst. Sci.*, 69(2):244–258, 2004.

[BSS03] Howard Barnum, Michael Saks, and Mario Szegedy. Quantum decision trees and semidefinite programming. In *Proc. 18th IEEE Complexity*, pages 179–193, 2003.

[CCJY07] Andrew M. Childs, Richard Cleve, Stephen P. Jordan, and David Yeung. Discrete-query quantum algorithm for NAND trees. arXiv:quant-ph/0702160, 2007.

[CEMM98] Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. *Proc. R. Soc. London A*, 454(1969):339–354, 1998.

[CF02] Ronald Cramer and Serge Fehr. Optimal black-box secret sharing over arbitrary Abelian groups. In *Proc. CRYPTO 2002*, LNCS vol. 2442, pages 272–287. Springer-Verlag, 2002.

[CRŠZ07] Andrew M. Childs, Ben W. Reichardt, Robert Špalek, and Shengyu Zhang. Every NAND formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. arXiv:quant-ph/0703015, 2007.

[FGG07] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum algorithm for the Hamiltonian NAND tree. arXiv:quant-ph/0702144, 2007.

[GP03] Anna Gál and Pavel Pudlák. A note on monotone complexity and the rank of matrices. *Information Processing Letters*, 87(6):321–326, 2003.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proc. 28th ACM STOC*, pages 212–219, 1996.

[Gro02] Lov K. Grover. Tradeoffs in the quantum search algorithm. arXiv:quant-ph/0201152, 2002.

[GV96] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins, Baltimore, 3rd edition, 1996.

[HLŠ05] Peter Hoyer, Troy Lee, and Robert Špalek. Tight adversary bounds for composite functions. arXiv:quant-ph/0509067, 2005.

[HLŠ06] Peter Hoyer, Troy Lee, and Robert Špalek. Source codes of semidefinite programs for $\text{ADV}^{\pm}$. `http://www.ucw.cz/~robert/papers/adv/`, 2006.

[HLŠ07] Peter Hoyer, Troy Lee, and Robert Špalek. Negative weights make adversaries stronger. In *Proc. 39th ACM STOC*, pages 526–535, 2007.

[JKS03] T. S. Jayram, Ravi Kumar, and D. Sivakumar. Two applications of information complexity. In *Proc. 35th ACM STOC*, pages 673–682, 2003.

[KW93] Mauricio Karchmer and Avi Wigderson. On span programs. In *Proc. 8th IEEE Symp. Structure in Complexity Theory*, pages 102–111, 1993.

[LM04] Sophie Laplante and Frédéric Magniez. Lower bounds for randomized and quantum query complexity using Kolmogorov arguments. In *Proc. 19th IEEE Complexity*, pages 294–304, 2004.

[MNRS07] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. In *Proc. 39th ACM STOC*, pages 575–584, 2007.

[NNP05] Ventzislav Nikov, Svetla Nikova, and Bart Preneel. On the size of monotone span programs. In *Proc. SCN 2004*, LNCS vol. 3352, pages 249–262, 2005.

[RŠ07] Ben W. Reichardt and Robert Špalek. Quantum query complexity of up to 4-bit functions. `http://www.ucw.cz/~robert/papers/gadgets/`, 2007.

[San95] Miklos Santha. On the Monte Carlo decision tree complexity of read-once formulae. *Random Structures and Algorithms*, 6(1):75–87, 1995. Preliminary version in *Proc. 6th IEEE Structure in Complexity Theory*, 1991.

[Sni85] Marc Snir. Lower bounds on probabilistic linear decision trees. *Theoretical Computer Science*, 38:69–82, 1985.

[ŠS06] Robert Špalek and Mario Szegedy. All quantum adversary methods are equivalent. *Theory of Computing*, 2(1):1–18, 2006. Earlier version in ICALP'05.

[SW86] Michael Saks and Avi Wigderson. Probabilistic Boolean decision trees and the complexity of evaluating game trees. In *Proc. 27th IEEE FOCS*, pages 29–38, 1986.

[Sze04] Mario Szegedy. Quantum speed-up of Markov chain based algorithms. In *Proc. 45th IEEE FOCS*, pages 32–41, 2004.

A Proof of Theorem 3.7

In this section we will prove Theorem 3.7. Before beginning, though, let us show that the alternative expressions for the span program witness size $\mathrm{wsize}(P,x)$ of Definition 3.6 are equivalent, so $\mathrm{wsize}(P)$ is well defined. It will also be useful to derive several alternative expressions for $\mathrm{wsize}(P,x)$.

Definition A.1 (Additional matrix notations)**.** *Recall Definition 3.4. Let $\langle o| = A_{OJ} = \langle t|A$ and let $C = (1-|t\rangle\langle t|)A$, so $A = C + |t\rangle\langle o|$ (the matrix A_{CJ} is C with range restricted to $1-|t\rangle\langle t|$). Let $\Delta = C\Pi(C\Pi)^+$ be the projection onto the range of $C\Pi$, and let $\overline{\Delta} = 1-\Delta$. For a matrix M and a projection Θ, let M_Θ denote the restriction $\Theta M\Theta$ of M to the range of Θ. For a vector $\tilde{m}_J$, we will commonly write $\tilde{m} = \sum_{j\in J}\tilde{m}_j|j\rangle\langle j|$ for the diagonal matrix with diagonal entries $\tilde{m}_J$. Figure 6 summarizes the matrices used in this section.*

We will use several times the following estimates for pseudoinverse norms:

Claim A.2. *For matrices A and B with $\mathrm{Range}(B^\dagger) \subseteq \mathrm{Range}(A)$ (i.e., $B = BAA^+$), $\|A(BA)^+\| \leq \|B^+\|$ and $\|(BA)^+\| \leq \|A^+\|\|B^+\|$.*

Proof. Since $A(BA)^+ = A(BA)^+BB^+ = \big[A(BA)^+(BA)A^+\big]B^+$ and the bracketed term is a projection, $\|A(BA)^+\| \leq \|B^+\|$. Then also $\|(BA)^+\| = \|A^+A(BA)^+\| \leq \|A^+\|\|B^+\|$. □

Lemma A.3. *For S any positive-definite, diagonal matrix, let*

$$\mathrm{wsize}_S(P,x) = \begin{cases} \min_{|w\rangle:A\Pi|w\rangle=|t\rangle} \||S|w\rangle\|^2 & \text{if } f_P(x)=1 \\ \min_{\substack{|w'\rangle:\langle t|w'\rangle=1 \\ \Pi A^\dagger|w'\rangle=0}} \||SA^\dagger|w'\rangle\|^2 & \text{if } f_P(x)=0 \end{cases} \tag{A.1}$$

Then if $f_P(x)=1$,

$$\begin{aligned} \mathrm{wsize}_S(P,x) &= \min_{\substack{|w\rangle:\langle w|\Pi|w\rangle=1 \\ C\Pi S^{-1}|w\rangle=0}} |\langle o|\Pi S^{-1}|w\rangle|^{-2} \\ &= \|(A\Pi S^{-1})^+|t\rangle\|^2 \\ &= \|(\Pi - (C\Pi S^{-1})^+C\Pi S^{-1})S^{-1}|o\rangle\|^{-2} \end{aligned} \tag{A.2}$$

and, if $f_P(x)=0$,

$$\begin{aligned} \mathrm{wsize}_S(P,x) &= \min_{\substack{|w'\rangle:\||SA^\dagger|w'\rangle\|=1 \\ \Pi A^\dagger|w'\rangle=0}} |\langle t|w'\rangle|^{-2} \\ &= \|\big(1+(\overline{\Pi}(AS)^+AS-1)^+\Pi\big)(AS)^+|t\rangle\|^{-2} \\ &= \|(1-(\overline{\Delta}CS)^+\overline{\Delta}CS)S(1-C^\dagger(\Pi C^\dagger)^+)|o\rangle\|^2 \ . \end{aligned} \tag{A.3}$$

Moreover, $|w_S^*\rangle = \arg\min_{|w\rangle:A\Pi|w\rangle=|t\rangle} \|S|w\rangle\|^2 = S^{-1}(A\Pi S^{-1})^+|t\rangle$ *has norm* $\||w_S^*\rangle\| = O(1)$ *and*

$$|w_S'^*\rangle = \arg\min_{\substack{|w'\rangle:\langle t|w'\rangle=1 \\ \Pi A^\dagger|w'\rangle=0}} \|SA^\dagger|w'\rangle\|^2 = |t\rangle - (\Pi C^\dagger)^+\Pi|o\rangle - (SC^\dagger\overline{\Delta})^+S(1-C^\dagger(\Pi C^\dagger)^+)|o\rangle$$

has norm $\||w_S'^*\rangle\| = O(1)$.

In particular, the two different expressions for $\mathrm{wsize}(P,x) = \mathrm{wsize}_{\sqrt{\tilde{z}}}(P,x)$ *in Definition 3.6 are equivalent, so* $\mathrm{wsize}(P,x)$ *and* $\mathrm{wsize}(P)$ *are well defined.*

Proof. Assume $f_P(x) = 1$. That $\min_{|w\rangle:A\Pi S^{-1}|w\rangle=|t\rangle} \||w\rangle\|^2 = \min_{\substack{|w\rangle:\langle w|\Pi|w\rangle=1 \\ C\Pi S^{-1}|w\rangle=0}} |\langle o|\Pi S^{-1}|w\rangle|^{-2}$ is immediate. In general, $\arg\min_{|x\rangle:M|x\rangle=|b\rangle} \||x\rangle\| = M^+|b\rangle$; therefore, $\min_{|w\rangle:A\Pi S^{-1}|w\rangle=|t\rangle} \||w\rangle\|^2 = \|(A\Pi S^{-1})^+|t\rangle\|^2$. By basic geometry, $\arg\max_{\substack{|w\rangle:\|\Pi|w\rangle\|=1 \\ C\Pi S^{-1}|w\rangle=0}} |\langle o|\Pi S^{-1}|w\rangle| \propto \big(1-(C\Pi S^{-1})^+(C\Pi S^{-1})\big)\Pi S^{-1}|o\rangle$, i.e., is proportional to the projection of $\Pi S^{-1}|o\rangle$ onto the space orthogonal to the range of $S^{-1}\Pi C^\dagger$. Eq. (A.2) follows.

Next assume $f_P(x) = 0$. That $\min_{\substack{|w'\rangle:\langle t|w'\rangle=1 \\ \Pi A^\dagger|w'\rangle=0}} \|SA^\dagger|w'\rangle\|^2 = \min_{\substack{|w'\rangle:\|SA^\dagger|w'\rangle\|=1 \\ \Pi A^\dagger|w'\rangle=0}} |\langle t|w'\rangle|^{-2}$ is immediate. Now, without loss of generality, $|t\rangle \in \mathrm{Range}(A) = \mathrm{Range}(AS)$, since otherwise f_P is false on every input. Therefore, $\langle t|w'\rangle = \langle t|(SA^\dagger)^+(SA^\dagger)|w'\rangle = \langle t|(SA^\dagger)^+|w\rangle$ if $|w\rangle = SA^\dagger|w'\rangle$. We want to find the length-one vector $|w\rangle$ that is in the range of $SA^\dagger$ and also of $\overline{\Pi}$, and that maximizes $|\langle t|(SA^\dagger)^+|w\rangle|^2$. The answer is clearly the normalized projection of $(AS)^+|t\rangle$ onto the intersection $\mathrm{Range}(SA^\dagger) \cap \mathrm{Range}(\overline{\Pi})$. In general, given two projections Π_1 and Π_2, the projection onto the intersection of their ranges can be written $1 - (\Pi_1\Pi_2 - 1)^+(\Pi_1\Pi_2 - 1)$. Substituting $\Pi_1 = \overline{\Pi}$ and $\Pi_2 = (AS)^+AS$ gives the second claimed expression.

Finally, we show that $\min_{\substack{|w'\rangle:\langle t|w'\rangle=1 \\ \Pi A^\dagger|w'\rangle=0}} \|SA^\dagger|w'\rangle\|^2 = \|(1-(\overline{\Delta}CS)^+\overline{\Delta}CS)S(1-C^\dagger(\Pi C^\dagger)^+)|o\rangle\|^2$. Since $f_P(x)$ is false, $|t\rangle$ does not lie in the span of the true grouped input vectors, $|t\rangle \notin \mathrm{Range}(A\Pi)$, or equivalently $\Pi|o\rangle \in \mathrm{Range}(\Pi C^\dagger)$. Therefore, there exists a vector $|w'\rangle = |t\rangle + |b_C\rangle$ that is orthogonal to the span of the true columns of A and has inner product one with $|t\rangle$. Any such $|b_C\rangle$ has the form

$$|b_C\rangle = -(\Pi C^\dagger)^+\Pi|o\rangle + \overline{\Delta}|v\rangle \ ,$$

where $|v\rangle$ is an arbitrary vector with $\langle t|v\rangle = 0$. We want to choose $|v\rangle$ to minimize the squared length of

$$\begin{aligned}\overline{\Pi}SA^\dagger|w'\rangle &= S\overline{\Pi}(|o\rangle + C^\dagger|b_C\rangle) \\ &= S(1-C^\dagger(\Pi C^\dagger)^+)|o\rangle + SC^\dagger\overline{\Delta}|v\rangle \ . \end{aligned} \tag{A.4}$$

The answer is clearly the squared length of $S(1-C^\dagger(\Pi C^\dagger)^+)|o\rangle$ projected orthogonal to the range of $SC^\dagger\overline{\Delta}$, as claimed. This corresponds to setting $|v\rangle = -(SC^\dagger\overline{\Delta})^+S(1-C^\dagger(\Pi C^\dagger)^+)|o\rangle$.

The norms of $|w_S^*\rangle$ and $|w_S'^*\rangle$ are bounded using Claim A.2. □

Remark A.4. *The expressions for witness size in Eqs. (A.2) and (A.3) look quite different depending on whether* $f_P(x) = 1$ *or* $f_P(x) = 0$*, with the latter case being more complicated. It can be seen, though, that* $|\mathrm{wsize}(P,x) - \mathrm{wsize}(P^\dagger,x)| = O(1)$ *for any fixed span program* P*, where* $P^\dagger$ *is the dual span program described in Section 2.3 with* $f_{P^\dagger}(x) = \neg f_P(x)$.

Let us now show that if $\tilde{z}'_j \lesssim \tilde{z}_j$ for all $j \in J$, then $\mathrm{wsize}_{\sqrt{\tilde{z}'}}(P,x) \lesssim \mathrm{wsize}(P,x)$ (Lemma A.6). This will be useful in showing that $\mathrm{wsize}(P,x)$ is a rough upper bound on the exact expressions that we will derive in the sections below.

Remark A.5. *From Definition 3.6, it is immediate that* $\mathrm{wsize}(P,x)$ *is monotone increasing in each input complexity* z_i.

Lemma A.6. *Let* S *and* T *be any positive-definite diagonal matrices. Then*

$$\mathrm{wsize}_{S\sqrt{1+T}}(P,x) \le \mathrm{wsize}_S(P,x)\cdot(1+\|T\|) \ , \tag{A.5}$$

$$\mathrm{wsize}_{\sqrt{S^2+T^2}}(P,x) \le \mathrm{wsize}_S(P,x) + O(\|T\|^2) \ . \tag{A.6}$$

In particular, if $\tilde{z}'_J$ *is such that* $\tilde{z}'_j \lesssim \tilde{z}_j$ *for all* $j \in J$*, then* $\mathrm{wsize}_{\sqrt{\tilde{z}'}}(P,x) \lesssim \mathrm{wsize}_{\sqrt{\tilde{z}}}(P,x) = \mathrm{wsize}(P,x)$.

$A = \sum_j \|v_j\rangle\langle j\| = C + \|t\rangle\langle o\|$	span program matrix
$C = (1 - \|t\rangle\langle t\|)A$	"constraint" part of the span program
$\langle o\| = A_{OJ} = \langle t\|A$	"output" row of the span program
$\Pi = \sum_{\text{true } j} \|j\rangle\langle j\|$	projection onto true grouped inputs
$\overline{\Pi} = 1 - \Pi$	projection onto false grouped inputs
$\tilde{\sigma} = \sum_j \tilde{\sigma}_j \|j\rangle\langle j\|$	grouped input squared supports at $\lambda = 0$
$r = \sum_i r_i \|i\rangle\langle i\|$	input ratios $r_i = a_{j(i)}/b_i$
$\tilde{r} = (A_{IJ}{}^\dagger r^{-1} A_{IJ} - \lambda)^{-1} = \sum_j \tilde{r}_j \|j\rangle\langle j\|$	grouped input ratios
$\tilde{s} = -\frac{1}{\lambda}\Pi\tilde{r}^{-1} + \frac{1}{\lambda}\overline{\Pi}\tilde{r} = \sum_j \tilde{s}_j \|j\rangle\langle j\|$	grouped input ratio multipliers
$\tilde{z} = \sum_j \tilde{z}_j \|j\rangle\langle j\|$	grouped input complexities, $\frac{1}{\tilde{\sigma}_j}, \tilde{s}_j \lesssim \tilde{z}_j$
$y = C\tilde{s}^{-1/2}\Pi$, $Y = yy^\dagger$	true constraints scaled down by $\sqrt{\tilde{s}}$
$\bar{y} = C\tilde{s}^{1/2}\overline{\Pi}$, $\bar{Y} = \bar{y}\bar{y}^\dagger$	false constraints scaled up by $\sqrt{\tilde{s}}$
$\Delta = yy^+ = C\Pi(C\Pi)^+$	projection onto the range of true constraints
$\overline{\Delta} = 1 - \Delta$	complementary projection
$X = -\lambda^2(1 + \frac{1}{\lambda}C\tilde{r}C^\dagger) = Y - \lambda^2(\bar{Y} + 1)$	matrix to be inverted
$(X/X_\Delta) = X_{\overline{\Delta}} - \overline{\Delta}X(X_\Delta)^{-1}X\overline{\Delta}$	Schur complement of X_Δ in X
$V = -\lambda^2\bar{y}^\dagger\overline{\Delta}X^{-1}\overline{\Delta}\bar{y} = -\lambda^2\bar{y}^\dagger(X/X_\Delta)^{-1}\bar{y}$	a part of the inverse of X
$\widehat{V} = \bar{y}^\dagger\overline{\Delta}(\bar{Y}_{\overline{\Delta}} + 1)^{-1}\overline{\Delta}\bar{y}$	a useful $O(1)$ matrix, $V - \widehat{V} = O(\lambda^2\|\tilde{z}\|^2)$

Figure 6: Matrices used in the proof of Theorem 3.7.

Proof. Eq. (A.5) is immediate from the definition in Eq. (A.1). To derive Eq. (A.6), first note that $\|\sqrt{S^2 + T^2}|v\rangle\|^2 = \|S|v\rangle\|^2 + \|T|v\rangle\|^2$. Then when $f_P(x) = 1$,

$$\begin{aligned}\min_{|w\rangle : A\Pi|w\rangle = |t\rangle} (\|S|w\rangle\|^2 + \|T|w\rangle\|^2) &\le \|S|w_S^*\rangle\|^2 + \|T|w_S^*\rangle\|^2 \\ &= \text{wsize}_S(P, x) + O(\|T\|^2) \ ,\end{aligned}$$

where we have used that $|w_S^*\rangle = \arg\min_{|w\rangle : A\Pi|w\rangle = |t\rangle} \|S|w\rangle\|^2$ has norm $\||w_S^*\rangle\| = O(1)$ by Lemma A.3. The argument when $f_P(x) = 0$ is similar:

$$\min_{\substack{|w'\rangle : \langle t|w'\rangle = 1 \\ \Pi A^\dagger|w'\rangle = 0}} (\|SA^\dagger|w'\rangle\|^2 + \|TA^\dagger|w'\rangle\|^2) \le \text{wsize}_S(P, x) + \|TA^\dagger|w_S'^*\rangle\|^2 \ ,$$

where $|w_S'^*\rangle = \arg\min_{\substack{|w'\rangle : \langle t|w'\rangle = 1 \\ \Pi A^\dagger|w'\rangle = 0}} \|SA^\dagger|w'\rangle\|^2$ has $O(1)$ norm by Lemma A.3. □

A.1 Quantitative eigenvalue-zero spectral analysis of A_{G_P}

Theorem 2.5 can be strengthened to put quantitative lower bounds on σ_O, the achievable squared magnitude, in a unit-normalized eigenvalue-zero eigenvector, on the output node either a_O if $f_P(x) = 1$ or b_O if $f_P(x) = 0$:

Theorem A.7. *For an input $x \in \{0,1\}^n$, define a weighted graph $G_P(x)$ by deleting from G_P the edges (a_j, b_i) if the ith literal in X_j is true. Also let*

$$W = 1 + \sum_{j \in J} \sum_{i \in I_j} \left(\frac{1}{\sigma_i} - 1\right) (|b_i\rangle\langle b_i| + |a_j\rangle\langle a_j|) \ . \tag{A.7}$$

Consider all the eigenvalue-zero eigenvector equations of the weighted adjacency matrix $A_{G_P(x)}$, except for the constraint at a_O, i.e., Eqs. (2.1) except (2.1d). By Theorem 2.5, these equations have a solution $|\psi\rangle$ with

$\langle a_O|\psi\rangle \neq 0$ if and only if $f_P(x) = 1$, and have a solution $|\psi\rangle$ with $\langle b_O|\psi\rangle \neq 0$ if and only if $f_P(x) = 0$. In fact, the solution $|\psi\rangle$ can be chosen so that the normalized square overlap

$$\sigma_O \equiv \frac{|(\langle a_O| + \langle b_O|)|\psi\rangle|^2}{\langle\psi|W|\psi\rangle} \geq \frac{1}{\text{wsize}(P,x) + \text{constant}} \ , \tag{A.8}$$

where the constant may depend on P but is independent of the σ_I, and $\text{wsize}(P,x)$ *is as defined in Definition 3.6, with $1/\sigma_i \leq z_i$ for all i.*

Remark A.8. *Note that Theorem A.7 implies the $\lambda = 0$ portion of Theorem 3.7. The weights in W mean that, e.g., setting $b_i = 1$ adds $\langle b_i|W|b_i\rangle = 1/\sigma_i$ to the squared normalization factor.*

Proof of Theorem A.7. Recall Figure 2. The vertex a_j is a shared output node of all the inputs $i \in I_j$. As in the proof of Theorem 2.5, Eq. (2.1c) implies that a_j can be nonzero only if grouped input j is true, i.e., if all $i \in I_j$ evaluate to true.

For $j \in J$, define $\tilde{\sigma}_j$ by

$$\tilde{\sigma}_j = \begin{cases} \left(1 + \sum_{i\in I_j} \left(\frac{1}{\sigma_i} - 1\right)\right)^{-1} & \text{if } j \text{ is true} \\ \sum_{\text{false } i \in I_j} \sigma_i & \text{if } j \text{ is false} \end{cases} \tag{A.9}$$

From Definition 3.1, $1/\tilde{\sigma}_j \leq \tilde{z}_j$. Roughly speaking, for each j, the vertices b_i for $i \in I_j$ can be treated as just a single input vertex with associated weight $\tilde{\sigma}_j$ in W. Precisely, if j is true, then $\langle a_j|W|a_j\rangle = 1/\tilde{\sigma}_j$. And if j is false, then the b_{I_j} coefficients appear in Eq. (2.1e) only in the quantity $\langle j|A_{IJ}(x)^\dagger|b_{I_j}\rangle = \sum_{\text{false } i \in I_j} b_i$. In order to minimize the weighted squared norm $\langle b_{I_j}|W|b_{I_j}\rangle = \sum_{i\in I_j}|b_i|^2/\sigma_i$ for any fixed value of $\langle j|A_{IJ}(x)^\dagger|b_{I_j}\rangle$, each b_i for i false should be set proportional to σ_i (by Cauchy-Schwarz), so

$$\min_{|b_{I_j}\rangle:\langle j|A_{IJ}(x)^\dagger|b_{I_j}\rangle=1} \langle b_{I_j}|W|b_{I_j}\rangle = \frac{1}{\tilde{\sigma}_j} \ . \tag{A.10}$$

Let $\tilde{\sigma} = \sum_j \tilde{\sigma}_j |j\rangle\langle j|$.

Case $f_P(x) = 1$: When $f_P(x) = 1$, set $a_j = 0$ for all false grouped inputs j. Set the other a_j so as to maximize the magnitude of $-a_O = A_{OJ}|a_J\rangle = \langle o|\Pi|a_J\rangle$, such that $C|a_J\rangle = C\Pi|a_J\rangle = 0$ and $\langle a_J|W|a_J\rangle = \langle a_J|\Pi\tilde{\sigma}^{-1}|a_J\rangle = 1$ (Eqs. (2.1b) and (2.1a) at $\lambda = 0$). Now, changing variables to $|w\rangle = \tilde{\sigma}^{-1/2}|a_J\rangle$,

$$\begin{aligned} |a_O|^2 &= \max_{\substack{|w\rangle:\langle w|\Pi|w\rangle=1 \\ C\Pi\tilde{\sigma}^{1/2}|w\rangle=0}} |\langle o|\Pi\tilde{\sigma}^{1/2}|w\rangle|^2 \\ &= 1/\text{wsize}_{\tilde{\sigma}^{-1/2}}(P,x) \\ &\geq 1/\text{wsize}(P,x) \ , \end{aligned} \tag{A.11}$$

using Eq. (A.2), $1/\tilde{\sigma}_j \leq \tilde{z}_j$ and the monotonicity of $\text{wsize}(P,x)$ (Remark A.5). Finally, dividing by $(1 + |a_O|^2)$ so that the total norm is one, gives

$$\sigma_O = \frac{|a_O|^2}{1 + |a_O|^2} \geq \frac{1}{\text{wsize}(P,x) + 1} \ . \tag{A.12}$$

Case $f_P(x) = 0$: When $f_P(x) = 0$, for each true grouped input j set $b_i = 0$ for $i \in I_j$. For each false j, set $b_i = 0$ for true $i \in I_j$ and set $b_i = f_j\sigma_i / \sum_{\text{false } i' \in I_j} \sigma_{i'}$. Choose $|f\rangle$ to maximize $|b_O|^2$ such that $\langle f|\overline{\Pi}\tilde{\sigma}^{-1}|f\rangle = 1$ and, by Eq. (2.1e) at $\lambda = 0$, $|o\rangle b_O + C^\dagger|b_C\rangle + \overline{\Pi}|f\rangle = 0$. Equivalently, writing

$|w'\rangle = \binom{b_O}{b_C}$ so $b_O = \langle t|w'\rangle$, we are constrained that $\overline{\Pi}|f\rangle = -A^\dagger|w'\rangle$, i.e.,

$$\begin{aligned} |b_O|^2 &= \max_{\substack{|w'\rangle:\|\tilde{\sigma}^{-1/2}A^\dagger|w'\rangle\|=1 \\ \Pi A^\dagger|w'\rangle=0}} |\langle t|w'\rangle|^2 \\ &= 1/\mathrm{wsize}_{\tilde{\sigma}^{-1/2}}(P,x) \\ &\geq 1/\mathrm{wsize}(P,x) \end{aligned} \tag{A.13}$$

by Eq. (A.3) and the monotonicity of $\mathrm{wsize}(P,x)$. The constructed state has weighted squared norm $\langle\psi|W|\psi\rangle = 1 + \||w'^*\rangle\|^2$, where $|w'^*\rangle = \arg\max_{\substack{|w'\rangle:\|\tilde{\sigma}^{-1/2}A^\dagger|w'\rangle\|=1 \\ \Pi A^\dagger|w'\rangle=0}} |\langle t|w'\rangle|^2$. Normalizing,

$$\begin{aligned} \frac{1}{\sigma_O} &= \frac{1 + \||w'^*\rangle\|^2}{|b_O|^2} \\ &\leq \mathrm{wsize}(P,x) + \frac{\||w'^*\rangle\|^2}{|b_O|^2} \ . \end{aligned} \tag{A.14}$$

It remains to show that $\||w'^*\rangle/b_O\| = O(1)$. Indeed, $\frac{1}{\langle t|w'^*\rangle}|w'^*\rangle = \arg\min_{\substack{|w'\rangle:\langle t|w'\rangle=1 \\ \Pi A^\dagger|w'\rangle=0}} \|\tilde{\sigma}^{-1/2}A^\dagger|w'\rangle\|^2 = |w'^*_{\tilde{\sigma}^{-1/2}}\rangle$ has $O(1)$ norm by Lemma A.3. □

A.2 Small-eigenvalue spectral analysis of A_{G_P}

Theorem A.9. *For a span program P and input x, given s_I with $0 < s_i \leq z_i$ for all $i \in I$, let $s = \sum_i s_i|i\rangle\langle i|$ and $r = \sum_i r_i|i\rangle\langle i| = -\frac{1}{\lambda}\Pi s^{-1} + \lambda\overline{\Pi}s$. Assume that $0 < \lambda \leq \epsilon/s_i$ for a small enough constant $\epsilon > 0$ to be determined and for all $i \in I$. Then the equations*

$$b_I = r^{-1}A_{IJ}a_J \tag{A.15a}$$
$$\lambda b_O = a_O + A_{OJ}a_J \tag{A.15b}$$
$$\lambda b_C = A_{CJ}a_J \tag{A.15c}$$
$$\lambda a_J = A_{OJ}{}^\dagger b_O + A_{CJ}{}^\dagger b_C + A_{IJ}(x)^\dagger b_I \tag{A.15d}$$

have a solution with $a_O, b_O \neq 0$. Moreover, if $r_O = a_O/b_O$ and s_O is defined as $-1/(\lambda r_O)$ or r_O/λ if $f_P(x)$ is true or false, respectively, then

$$0 < s_O \lesssim \mathrm{wsize}_{\sqrt{\tilde{z}}}(P,x) \ , \tag{A.16}$$

where the grouped input complexities $\tilde{z}_J$ are defined in terms of z_I in Definition 3.2.

Proof. Similarly to the argument in Appendix A.1, it will be useful to define a "grouped input ratio" $\tilde{r}_j$ so that, roughly speaking, the vertices b_i for $i \in I_j$ can be treated as just a single input vertex.

Definition A.10 (Grouped input ratios)**.** *For $j \in J$, let $\tilde{r}_j = (-\lambda + \sum_{i\in I_j} r_i^{-1})^{-1}$, and let $\tilde{r} = \sum_j \tilde{r}_j|j\rangle\langle j| = (A_{IJ}{}^\dagger r^{-1}A_{IJ} - \lambda)^{-1}$. Like an input ratio r_i, $\tilde{r}_j$ is large and negative if j is true, and small and positive if j is false. Therefore let $\tilde{s}_j = -1/(\lambda\tilde{r}_j)$ if j is true, and $\tilde{s}_j = \tilde{r}_j/\lambda$ if j is false. Let $\tilde{s} = \sum_j \tilde{s}_j|j\rangle\langle j| = -\frac{1}{\lambda}\Pi\tilde{r}^{-1} + \frac{1}{\lambda}\overline{\Pi}\tilde{r}$, so $\tilde{r} = -\frac{1}{\lambda}\Pi\tilde{s}^{-1} + \lambda\overline{\Pi}\tilde{s}$.*

Before proceeding, we need to establish that $\tilde{r}$ and $\tilde{s}$ are well defined.

Lemma A.11. *Assume that $0 < \lambda \leq \epsilon/z_i$ for a small enough constant $\epsilon > 0$ and for all $i \in I$. Then $\tilde{r} = (A_{IJ}{}^\dagger r^{-1}A_{IJ} - \lambda)^{-1}$ exists, so $\tilde{s}$ exists as well. Moreover, for each grouped input $j \in J$, $\tilde{s}_j \lesssim \tilde{z}_j$.*

Proof. By definition,

$$\tilde{r}_j = (-\lambda + \sum_{i \in I_j} r_i^{-1})^{-1} = \Big(-\lambda - \lambda \sum_{\text{true } i \in I_j} s_i + \frac{1}{\lambda} \sum_{\text{false } i \in I_j} s_i^{-1} \Big)^{-1} \ .$$

If all inputs in I_j are true, then

$$\tilde{s}_j = -1/(\lambda \tilde{r}_j) = 1 + \sum_{i \in I_j} s_i \ ,$$

so $1 \le \tilde{s}_j \le 1 + \sum_{i \in I_j} z_i \le 1 + \tilde{z}_j$.

Now assume at least one input in I_j is false. The true terms can be upper-bounded by $\lambda \sum_{\text{true } i \in I_j} s_i \le \lambda \sum_{i \in I_j} z_i \le |I_j|\epsilon$. On the other hand, if i is false then $(\lambda s_i)^{-1} \ge (\lambda z_i)^{-1} \ge 1/\epsilon$. Therefore, $\tilde{s}_j > 0$, and we also get $\tilde{s}_j \le \tilde{z}_j(1 + \epsilon' \lambda \max_{\text{false } i \in I_j} z_i)$ for a constant ϵ'. □

Now we will solve for the output ratio r_O using Eqs. (A.15b-d). Letting $s_O = r_O/\lambda$ in case $f_P(x) = 0$, or $s_O = -1/(\lambda r_O)$ in case $f_P(x) = 1$, we aim to show that $0 < s_O \lesssim \text{wsize}_{\sqrt{\tilde{s}}}(P, x)$. This will prove Theorem A.9 since, by Lemma A.6, $\text{wsize}_{\sqrt{\tilde{s}}}(P, x) \lesssim \text{wsize}_{\sqrt{\tilde{z}}}(P, x) = \text{wsize}(P, x)$. Our proof will follow the sketch below Theorem 3.7 in Section 3. We start by deriving an exact expression for r_O:

Lemma A.12. *The solution to Eq. (A.15) has $a_O = 0$ if $b_O = 0$, and otherwise,*

$$r_O = \lambda + \langle o | \left(\tilde{r} - \frac{1}{\lambda} \tilde{r} C^\dagger (1 + \tfrac{1}{\lambda} C \tilde{r} C^\dagger)^{-1} C \tilde{r} \right) |o\rangle \ , \tag{A.17}$$

provided that $\tilde{r}$ and $(1 + \frac{1}{\lambda} C \tilde{r} C^\dagger)^{-1}$ exist.

Proof. Recall from Definition A.1 that $|o\rangle = A_{OJ}{}^\dagger$ and A_{CJ} is C with range restricted. Substituting Eqs. (A.15a) and (A.15c) into (A.15d), and rearranging terms gives

$$\left(\lambda - A_{IJ}{}^\dagger r^{-1} A_{IJ} - \frac{1}{\lambda} C^\dagger C \right) |a_J\rangle = |o\rangle b_O \ .$$

From Eq. (A.15b), if $b_O \neq 0$, then $a_O/b_O = \lambda - \langle o | a_J \rangle / b_O$, so

$$\begin{aligned} r_O &= \lambda + \langle o | (\tilde{r}^{-1} + \frac{1}{\lambda} C^\dagger C)^{-1} | o \rangle \\ &= \lambda + \langle o | \left(\tilde{r} - \frac{1}{\lambda} \tilde{r} C^\dagger (1 + \tfrac{1}{\lambda} C \tilde{r} C^\dagger)^{-1} C \tilde{r} \right) |o\rangle \ , \end{aligned} \tag{A.18}$$

by the Woodbury matrix identity [GV96], provided that $\tilde{r}$ and $(1 + \frac{1}{\lambda} C \tilde{r} C^\dagger)^{-1}$ exist. □

Remark A.13 (Form of Eq. (A.17))**.** *Note from Eq. (A.17) that r_O is a real number provided that all the input ratios r_I are themselves reals. Also, note that r_O depends on C only through $C^\dagger C$ (see too Eq. (A.18) in the proof); in particular, left-multiplying C by U where U is any linear isometry (i.e., satisfying $U^\dagger U = 1$) has no effect. Since the grouped input vectors v_J can be arbitrary in Definition 2.1, $C^\dagger C$ is in general an arbitrary $|J| \times |J|$ positive semidefinite matrix.*

Now the main step in simplifying Eq. (A.17) is dividing the matrix we want to invert into a 2×2 block matrix and applying the following well-known claim:

Claim A.14. *Let X be an operator, and let Δ and $\overline{\Delta} = 1 - \Delta$ be a projection and its complement. Assume that X_Δ is invertible. Let the "Schur complement" of X be $(X/X_\Delta) \equiv X_{\overline{\Delta}} - \overline{\Delta} X (X_\Delta)^{-1} X \overline{\Delta}$. If the Schur complement of X is invertible on $\overline{\Delta}$, then X is invertible, and X^{-1} is given by:*

$$\begin{aligned} \Delta X^{-1} \Delta &= (X_\Delta)^{-1} + (X_\Delta)^{-1} X (X/X_\Delta)^{-1} X (X_\Delta)^{-1} & \Delta X^{-1} \overline{\Delta} &= -(X_\Delta)^{-1} X (X/X_\Delta)^{-1} \\ \overline{\Delta} X^{-1} \Delta &= -(X/X_\Delta)^{-1} X (X_\Delta)^{-1} & \overline{\Delta} X^{-1} \overline{\Delta} &= (X/X_\Delta)^{-1} \end{aligned} \tag{A.19}$$

Proof. Multiply out the matrices. □

Lemma A.15. *The inverse* $(1+\frac{1}{\lambda}C\tilde{r}C^\dagger)^{-1}$ *exists, provided* $0 < \lambda \le \epsilon/z_i$ *for a small enough positive constant* ϵ *and for all* $i \in I$.

Proof. Let $y = C\tilde{s}^{-1/2}\Pi$, $\bar{y} = C\tilde{s}^{1/2}\overline{\Pi}$, $Y = yy^\dagger$ and $\bar{Y} = \bar{y}\bar{y}^\dagger$. We aim to show that X^{-1} exists, where

$$\begin{aligned} X &= -\lambda^2\big(1 + \frac{1}{\lambda}C\tilde{r}C^\dagger\big) \\ &= Y - \lambda^2(\bar{Y}+1) \ . \end{aligned} \tag{A.20}$$

Let $\Delta = yy^+$ be the projection onto the range of y, and let $\overline{\Delta} = 1 - \Delta$. Then

$$\begin{aligned} X_\Delta &= Y - \lambda^2\Delta(\bar{Y}+1)\Delta & \Delta X\overline{\Delta} &= -\lambda^2\Delta\bar{Y}\overline{\Delta} \\ \overline{\Delta}X\Delta &= -\lambda^2\overline{\Delta}\bar{Y}\Delta & X_{\overline{\Delta}} &= -\lambda^2\overline{\Delta}(\bar{Y}+1)\overline{\Delta} \end{aligned} \tag{A.21}$$

Now since Y is invertible on Δ (i.e., $\Delta = YY^+$), so is X. By the Neumann series,

$$(X_\Delta)^{-1} = Y^+(1 + O(\lambda^2\|\tilde{s}\|^2)) \ , \tag{A.22}$$

where we have used that $\|\bar{Y}\| = O(\|\tilde{s}\|)$ and $\|Y^+\| = O(\|\tilde{s}\|)$ (Claim A.2), and where we write $O(\lambda^2\|\tilde{s}\|^2)$ to mean some matrix with norm so-bounded. In particular, $(X_\Delta)^{-1}$ is positive definite on Δ.

Let (X/X_Δ) be the Schur complement of X_Δ in X,

$$\begin{aligned} (X/X_\Delta) &\equiv X_{\overline{\Delta}} - \overline{\Delta}X(X_\Delta)^{-1}X\overline{\Delta} \\ &= -\lambda^2\overline{\Delta}(\bar{Y}+1)\overline{\Delta} - \lambda^4\overline{\Delta}\bar{Y}(X_\Delta)^{-1}\bar{Y}\overline{\Delta} \ . \end{aligned} \tag{A.23}$$

As $-(X/X_\Delta)$ on $\overline{\Delta}$ is the sum of the positive definite matrix $\lambda^2\overline{\Delta}$ and positive semidefinite matrices, (X/X_Δ) is negative definite on $\overline{\Delta}$ and in particular is invertible on $\overline{\Delta}$.

Since X_Δ and (X/X_Δ) are each invertible, on Δ and on $\overline{\Delta}$, respectively, X^{-1} exists by Claim A.14, as claimed. □

The following discussion will use the notation from the proof of Lemma A.15. It will also be convenient to let $S = \sqrt{\tilde{s}}$. We have from Eq. (A.17)

$$\begin{aligned} r_O &= \lambda + \langle o|\tilde{r}|o\rangle + \lambda\langle o|\tilde{r}C^\dagger X^{-1}C\tilde{r}|o\rangle \\ &= \lambda - \frac{1}{\lambda}\langle o|S^{-2}\Pi|o\rangle + \lambda\langle o|S^2\overline{\Pi}|o\rangle + \begin{pmatrix} \frac{1}{\lambda}\langle o|S^{-1}y^\dagger X^{-1}yS^{-1}|o\rangle & +\lambda^3\langle o|S\bar{y}^\dagger X^{-1}\bar{y}S|o\rangle \\ -\lambda\big(\langle o|S^{-1}y^\dagger X^{-1}\bar{y}S|o\rangle & +\langle o|S\bar{y}^\dagger X^{-1}yS^{-1}|o\rangle\big) \end{pmatrix} \ . \end{aligned} \tag{A.24}$$

Our goal now is to expand the above expression as a series in λ, evaluating the coefficients of $1/\lambda$ and of λ, and bounding higher-order terms. In order to expand X^{-1} as a series, we use the block decomposition of X and Claim A.14.

Let us start by evaluating two expressions, $(X_\Delta)^{-1}$ and $V = -\lambda^2\bar{y}^\dagger(X/X_\Delta)^{-1}\bar{y}$, that will reappear frequently in the following analysis.

Claim A.16. $(X_\Delta)^{-1}$ *satisfies*

$$\begin{aligned} (X_\Delta)^{-1} &= Y^+ + \lambda^2 Y^+(\bar{Y}+1)Y^+ + Y^+O(\lambda^4\|\tilde{s}\|^3)Y^+ \\ &= Y^+(1 + O(\lambda^2\|\tilde{s}\|^2)) = O(\|\tilde{s}\|) \ . \end{aligned} \tag{A.25}$$

Proof. We know that $\|(X_\Delta)^{-1}\| = O(\|\tilde{s}\|)$. Note that for matrices A and B, $(A+B)^{-1} = (1 - (A+B)^{-1}B)A^{-1} = A^{-1}(1 - B(A+B)^{-1}) = A^{-1} - A^{-1}BA^{-1} + A^{-1}B(A+B)^{-1}BA^{-1}$ provided A and $A+B$ are invertible. Applying this with $A = Y$ and $B = -\lambda^2\Delta(\bar{Y}+1)\Delta$ gives

$$\begin{aligned} (X_\Delta)^{-1} &= Y^+ + \lambda^2 Y^+(\bar{Y}+1)Y^+ + \lambda^4 Y^+(\bar{Y}+1)(X_\Delta)^{-1}(\bar{Y}+1)Y^+ \\ &= Y^+ + \lambda^2 Y^+(\bar{Y}+1)Y^+ + Y^+O(\lambda^4\|\tilde{s}\|^3)Y^+ \ . \end{aligned}$$

□

Claim A.17. *Let* $V = -\lambda^2 \bar{y}^\dagger \overline{\Delta} X^{-1} \overline{\Delta} \bar{y} = -\lambda^2 \bar{y}^\dagger (X/X_\Delta)^{-1} \bar{y}$. *Then*

$$V = \widehat{V} + O(\lambda^2 \|\tilde{s}\|^2) \qquad \textit{where} \qquad \widehat{V} = \bar{y}^\dagger \overline{\Delta} (\bar{Y}_{\overline{\Delta}} + 1)^{-1} \overline{\Delta} \bar{y} \ . \tag{A.26}$$

In particular, $\|\widehat{V}\| < 1$ *and* $\|V\| < 1 + O(\lambda^2 \|\tilde{s}\|^2) = O(1)$.

Proof. We compute

$$\begin{aligned} V &= -\lambda^2 \bar{y}^\dagger (X_{\overline{\Delta}} - \overline{\Delta} X (X_\Delta)^{-1} X \overline{\Delta})^{-1} \bar{y} \\ &= \bar{y}^\dagger \left[\overline{\Delta}(\bar{Y} + 1)\overline{\Delta} + \lambda^2 \overline{\Delta} \bar{Y} (X_\Delta)^{-1} \bar{Y} \overline{\Delta} \right]^{-1} \bar{y} \\ &= M^\dagger \left[M \big(1 + \lambda^2 \bar{y}^\dagger (X_\Delta)^{-1} \bar{y}\big) M^\dagger + 1 \right]^{-1} M \ , \end{aligned} \tag{A.27}$$

where $M = \overline{\Delta}\bar{y}$. Now $\lambda^2 \bar{y}^\dagger (X_\Delta)^{-1} \bar{y} = O(\lambda^2 \|\tilde{s}\|^2)$ since $\bar{y} = O(\|\sqrt{\tilde{s}}\|)$. Therefore, $M\big(1 + \lambda^2 \bar{y}^\dagger (X_\Delta)^{-1} \bar{y}\big) M^\dagger + 1$ is the sum of positive-definite and positive-semidefinite matrices, hence is invertible. Again use $(A+B)^{-1} = (1 - (A+B)^{-1}B)A^{-1}$, now with $A = MM^\dagger + 1 = \bar{Y}_{\overline{\Delta}} + 1$ and $B = \lambda^2 M \bar{y}^\dagger (X_\Delta)^{-1} \bar{y} M^\dagger$, to get

$$\begin{aligned} V &= \widehat{V} - V(\lambda^2 \bar{y}^\dagger (X_\Delta)^{-1} \bar{y}) \widehat{V} \\ &= \widehat{V} \cdot \left[1 + (\lambda^2 \bar{y}^\dagger (X_\Delta)^{-1} \bar{y}) \widehat{V} \right]^{-1} \ , \end{aligned} \tag{A.28}$$

provided the inverse right-multiplying $\widehat{V}$ exists. Indeed, $M^\dagger (MM^\dagger + 1)^{-1} M < 1$ for an arbitrary matrix M and in particular for $M = \overline{\Delta}\bar{y}$ (if the singular-value decomposition of M is $M = \sum_i m_i |i\rangle\langle i'|$, then $M^\dagger (MM^\dagger + 1)^{-1} M = \sum_i \frac{m_i^2}{m_i^2 + 1} |i'\rangle\langle i'|$). Therefore the inverse does exist, and we obtain Eq. (A.26). □

Now, using Claim A.16 and Claim A.17, we find

$$\begin{aligned} \Delta X^{-1} \Delta &= (X_\Delta)^{-1} + (X_\Delta)^{-1} X (X/X_\Delta)^{-1} X (X_\Delta)^{-1} \\ &= (X_\Delta)^{-1} - \lambda^2 (X_\Delta)^{-1} \bar{y} V \bar{y}^\dagger (X_\Delta)^{-1} \\ &= Y^+ + \lambda^2 Y^+ (\bar{Y} + 1) Y^+ + Y^+ O(\lambda^4 \|\tilde{s}\|^3) Y^+ \\ &\quad - \lambda^2 Y^+ (1 + O(\lambda^2 \|\tilde{s}\|^2)) \bar{y} \big(\widehat{V} + O(\lambda^2 \|\tilde{s}\|^2)\big) \bar{y}^\dagger (1 + O(\lambda^2 \|\tilde{s}\|^2)) Y^+ \\ &= Y^+ + \lambda^2 Y^+ \left(-\bar{y} \widehat{V} \bar{y}^\dagger + \bar{Y} + 1 + O(\lambda^2 \|\tilde{s}\|^3) \right) Y^+ \ , \end{aligned} \tag{A.29}$$

and

$$\begin{aligned} \Delta X^{-1} \overline{\Delta} \bar{y} &= \lambda^2 (X_\Delta)^{-1} \bar{Y} (X/X_\Delta)^{-1} \bar{y} \\ &= -(X_\Delta)^{-1} \bar{y} V \\ &= -Y^+ (1 + O(\lambda^2 \|\tilde{s}\|^2)) \bar{y} (\widehat{V} + O(\lambda^2 \|\tilde{s}\|^2)) \\ &= -Y^+ \bar{y} \widehat{V} + Y^+ \cdot O(\lambda^2 \|\tilde{s}\|^{5/2}) \ . \end{aligned} \tag{A.30}$$

In particular, $\|\Delta X^{-1} \Delta\| = O(\|\tilde{s}\|)$ and $\|\Delta X^{-1} \overline{\Delta} \bar{y}\| = O(\|\tilde{s}\|^{3/2})$.

Let us now substitute the expressions we have derived into Eq. (A.24) for r_O. Consider each of the terms involving X^{-1} separately. First of all,

$$\begin{aligned} \frac{1}{\lambda} \langle o | S^{-1} y^\dagger X^{-1} y S^{-1} | o \rangle &= \frac{1}{\lambda} \langle o | S^{-1} y^\dagger (\Delta X^{-1} \Delta) y S^{-1} | o \rangle \\ &= \frac{1}{\lambda} \langle o | \Pi S^{-1} y^+ y S^{-1} \Pi | o \rangle \\ &\quad + \lambda \langle o | \Pi S^{-1} y^+ \big(1 + \bar{Y} - \bar{y} \widehat{V} \bar{y}^\dagger\big) (y^\dagger)^+ S^{-1} \Pi | o \rangle \\ &\quad + O(\lambda^3 \|\tilde{s}\|^3) \ , \end{aligned} \tag{A.31}$$

where we have substituted Eq. (A.29) and applied $\|S^{-1}y^\dagger Y^+\| = \|S^{-1}y^+\| = O(1)$ (Claim A.2).

Also, by Claim A.17 and Eqs. (A.29), (A.30),

$$\begin{aligned}\lambda^3\langle o|S\bar{y}^\dagger X^{-1}\bar{y}S|o\rangle &= \lambda^3\langle o|S\bar{y}^\dagger(\Delta+\overline{\Delta})X^{-1}(\Delta+\overline{\Delta})\bar{y}S|o\rangle \\ &= -\lambda\langle o|SVS|o\rangle + O(\lambda^3\|\tilde{s}\|^3) \\ &= -\lambda\langle o|\overline{\Pi}S\widehat{V}S\overline{\Pi}|o\rangle + O(\lambda^3\|\tilde{s}\|^3)\ .\end{aligned} \tag{A.32}$$

Lastly, by Eqs. (A.29), (A.30),

$$\begin{aligned}\lambda\langle o|S^{-1}y^\dagger X^{-1}\bar{y}S|o\rangle &= \lambda\langle o|S^{-1}y^\dagger\Delta X^{-1}(\Delta+\overline{\Delta})\bar{y}S|o\rangle \\ &= \lambda\langle o|S^{-1}y^\dagger Y^+(1+O(\lambda^2\|\tilde{s}\|^2))\bar{y}S|o\rangle \\ &\quad - \lambda\langle o|S^{-1}y^\dagger Y^+\left[\bar{y}\widehat{V}+O(\lambda^2\|\tilde{s}\|^{5/2})\right]S|o\rangle \\ &= \lambda\langle o|\Pi S^{-1}y^+(\bar{y}-\bar{y}\widehat{V})S\overline{\Pi}|o\rangle + O(\lambda^3\|\tilde{s}\|^3)\ .\end{aligned} \tag{A.33}$$

Substituting Eqs. (A.31), (A.32), (A.33) into the expression for r_O gives

$$r_O = -\frac{1}{\lambda}\|(1-y^+y)S^{-1}\Pi|o\rangle\|^2 + \lambda\langle v|\big(1-\widehat{V}\big)|v\rangle + O(\lambda+\lambda^3\|\tilde{s}\|^3)\ , \tag{A.34}$$

where $|v\rangle = (S\overline{\Pi} - \bar{y}^\dagger(y^\dagger)^+S^{-1})|o\rangle$. From the singular-value decomposition, one infers that $(1-M^\dagger(MM^\dagger+1)^{-1}M) - (1-M^+M) = M^+(1-(MM^\dagger+1)^{-1})(M^\dagger)^+$ for any matrix M, and in particular for $M = \overline{\Delta}\bar{y}$. Moreover, $\langle v|(\overline{\Delta}\bar{y})^+(1-(\bar{Y}_{\overline{\Delta}}+1)^{-1})((\overline{\Delta}\bar{y})^\dagger)^+|v\rangle = O(1)$, since $\|S(\overline{\Delta}\bar{y})^+\| = O(1)$ and $\|\langle v|S^{-1}\| = O(1)$. Therefore the above equation simplifies to

$$r_O = -\frac{1}{\lambda}\|(1-y^+y)S^{-1}\Pi|o\rangle\|^2 + \lambda\big(\|(1-(\overline{\Delta}\bar{y})^+(\overline{\Delta}\bar{y}))|v\rangle\|^2 + O(1)\big) + O(\lambda^3\|\tilde{s}\|^3)\ . \tag{A.35}$$

This is as far as we can simplify r_O in general. When $f_P(x) = 1$, the first term is $-1/(\lambda\,\mathrm{wsize}_S(P,x))$, as desired, using the last expression of Eq. (A.2) for $\mathrm{wsize}_S(P,x)$. Assume then that $f_P(x) = 0$, i.e., $\Pi|o\rangle \in \mathrm{Range}(\Pi C^\dagger)$. In this case, the first term in Eq. (A.35) is zero, and the second term can be simplified slightly further. Using $(y^\dagger)^+S^{-1}|o\rangle = (\Pi S^{-1}C^\dagger)^+S^{-1}\Pi|o\rangle = (\Pi C^\dagger)^+|o\rangle$ and $\Pi C^\dagger(\Pi C^\dagger)^+|o\rangle = \Pi|o\rangle$,

$$\begin{aligned}|v\rangle &= (S\overline{\Pi} - \bar{y}^\dagger(\Pi C^\dagger)^+)|o\rangle \\ &= (S\overline{\Pi} - \bar{y}^\dagger(\Pi C^\dagger)^+)|o\rangle + S\Pi(1-C^\dagger(\Pi C^\dagger)^+)|o\rangle \\ &= S(1-C^\dagger(\Pi C^\dagger)^+)|o\rangle\ .\end{aligned} \tag{A.36}$$

Moreover, since $\overline{\Delta}CS\Pi = 0$, $\overline{\Delta}\bar{y} = \overline{\Delta}CS\overline{\Pi} = \overline{\Delta}CS$. Therefore, $\|(1-(\overline{\Delta}\bar{y})^+(\overline{\Delta}\bar{y}))|v\rangle\|^2 = \mathrm{wsize}_S(P,x)$, as desired, using the last expression of Eq. (A.3) for $\mathrm{wsize}_S(P,x)$. This concludes the proof of Theorem A.9. □

Theorem A.9 completes the $\lambda \neq 0$ portion of Theorem 3.7, finishing its proof. □

B Quantum algorithm

The approach outlined in Section 4.4 is slightly indirect. To motivate it, we begin by briefly considering in Section B.1 a more direct algorithm $\mathcal{ALG}'$, that runs phase estimation directly on $\exp(iA_{\tilde{G}_P(x)})$. $\mathcal{ALG}'$ is analogous to the algorithm described by Cleve et al. [CCJY07] soon after the original NAND formula evaluation paper [FGG07]. Algorithm $\mathcal{ALG}'$ is nearly optimal, but not quite. The operator $\exp(iA_{\tilde{G}_P(x)})$ is a continuous-time quantum walk, and the overhead can be thought of as coming from simulating continuous-time quantum dynamics with a discrete computational model, in particular with discrete oracle queries. To avoid this overhead, the proof of Theorem 4.7 in Section B.2 works with a discrete-time quantum walk.

The approach in Section B.1 is optional motivation, and the reader may choose to skip directly to Section B.2.

B.1 Intuition: Continuous-time quantum walk algorithm

Theorem 4.16 immediately suggests the basic form of a quantum algorithm for evaluating $\varphi(x)$:

Algorithm $\mathcal{ALG}'$: Input $x \in \{0,1\}^N$, Output true/false.

1. Prepare an initial state on the output node, $|a_O\rangle$.
2. Run phase estimation, with precision $\delta_p \leq \frac{1}{\|A_{\tilde{G}_P(0^N)}\|} \frac{\epsilon}{\mathrm{Adv}(\varphi)}$ and small enough constant error rate δ_e, on the unitary $V = \exp(iA_{\tilde{G}_P(x)}/\|A_{\tilde{G}_P(0^N)}\|)$.
3. Output true if and only if the phase estimation output is $\lambda = 0$.

The idea of the second step is to "measure the Hamiltonian $A_{\tilde{G}_P(x)}$." In this step, we have normalized $A_{\tilde{G}_P(x)}$ by $\|A_{\tilde{G}_P(0^N)}\| \geq \|A_{\tilde{G}_P(x)}\|$ instead of by $\|A_{\tilde{G}_P(x)}\|$, in order to minimize dependence on the input x. This norm is $O(1)$ since the graph $\tilde{G}_P(0^N)$ has vertex degrees and edge weights all $O(1)$.

Algorithm $\mathcal{ALG}'$ evaluates $\varphi(x)$ correctly, with a constant gap between its completeness and soundness:

- Theorem 4.16 implies that if the formula evaluates to true, then $A_{\tilde{G}_P(x)}$ has an eigenvalue-zero eigenstate with squared support $|a_O|^2 = \Omega(1)$ on a_O. Therefore, the phase estimation outcome is $\lambda = 0$ with probability at least $|a_O|^2 - \delta_e = \Omega(1)$ (the completeness parameter).
- On the other hand, if the formula evaluates to false, then Theorem 4.16 implies that $A_{\tilde{G}_P(x)}/\|A_{\tilde{G}_P(0^N)}\|$ has no eigenvalue-λ eigenstates supported on a_O with $|\lambda| \leq \delta_p$. Therefore, the measured outcome will be $\lambda = 0$ only if there is an error in the phase estimation. By choosing δ_e a small enough constant, the soundness error δ_e will be bounded away from the completeness parameter.

The efficiency of $\mathcal{ALG}'$ also seems promising. Phase estimation of V with precision δ_p and error rate δ_e requires $O(1/(\delta_p\delta_e))$ calls to V [CEMM98]. Therefore, the second step requires only $O(\mathrm{Adv}(\varphi))$ calls to V. However, we still need to explain how to implement V. This is important because $A_{\tilde{G}_P(x)}$ depends on the input x. Therefore, implementing V requires querying the x. If each call to V requires many queries to the input oracle O_x of Eq. (4.3), then the overall query efficiency of $\mathcal{ALG}'$ will be poor.

Note now that only the input edges of $G_P(x)$ depend on the input x. Therefore, $A_{\tilde{G}_P(x)}$ can be split up into two terms: (input edges) + (all other edges). The first term can be exponentiated with only two queries to the input oracle O_x, while exponentiating the second term requires no input queries. The two terms do not commute, but the exponential of their sum can still be computed to sufficient precision by using a Lie product decomposition. These are more quantitative versions of identities like $e^{A+B} = \lim_{n\to\infty}(e^{A/n}e^{B/n})^n$. For more details, see [CCJY07].

Unfortunately, implementing the exponential of $A_{\tilde{G}_P(x)}$ will require $\omega(1)$ input queries. By using higher-order Lie product formulas, the overhead can be reduced to $\exp(O(\sqrt{\log|x|}))$, which is $N^{o(1)}$. However, this is still a super-constant overhead, so it appears that this approach cannot yield an optimal formula evaluation algorithm—the best we can hope for is $O(\mathrm{Adv}(\varphi)) \cdot N^{o(1)}$ queries.

B.2 Proof of Theorem 4.7: Discrete-time quantum walk algorithm

Therefore, we turn to the approach used in the NAND formula evaluation algorithm of [CRŠZ07]. Instead of running phase estimation on the exponential of $A_{\tilde{G}_P(x)}$, we construct a discrete-time, or "coined," quantum walk $U_x = \tilde{O}_x U_{0^N}$, where $\tilde{O}_x$ is the adjusted oracle of Eq. (4.9), that has spectrum and eigenvectors corresponding in a precise way to those of $A_{\tilde{G}_P(x)}$. Then we run phase estimation on U_x. Each call to U_x requires exactly one oracle query, so there is no query overhead.

B.2.1 Construction of the coined quantum walk U_x

The first step in constructing U_x is to decompose $\tilde{A}_{G_P(0^N)}$ into (constant)$\cdot \Delta^\dagger \circ \Delta$, where Δ is a square matrix with row norms one, and $\circ$ denotes the entrywise matrix product. We follow [CRŠZ07]. One minor technical difference, though, is that for us, $A_{\tilde{G}_P(x)}$ is a Hermitian matrix with possibly complex entries. In [CRŠZ07], the analogous weighted adjacency matrix, for the NAND formula φ, is a real symmetric matrix. Therefore, we need to slightly modify the construction of Δ to obtain the correct phases for the entries of $\tilde{A}_{G_P(0^N)}$.

Definition B.1. *For notational convenience, let $A = A_{\tilde{G}_P(0^N)}$ be the weighted adjacency matrix for $\tilde{G}_P(0^N)$. (Recall from Theorem 4.16 that $\tilde{G}_P(0^N)$ is the same as G_P except with the edge weight on the output edge (a_O, b_O) reduced.) $A = \sum_{v,w} A_{v,w}|v\rangle\langle w|$ is a Hermitian matrix.*

$\tilde{G}_P(0^N)$ is a bipartite graph, so we may color each vertex red or black, such that every edge is between one red vertex and one black vertex.

Claim B.2. *Let $A' = \sum_{v,w} |A_{v,w}||v\rangle\langle w|$ be the entrywise absolute value of A. A' is a real symmetric matrix. Let $\|A'\|$ be the largest-magnitude eigenvalue of A'. Let $|\delta\rangle$ be the principle eigenvector of A', with $\langle v|\delta\rangle = \delta_v > 0$ for every v, and let*

$$\Delta = \frac{1}{\sqrt{\|A'\|}} \sum_{\substack{black\ v \\ red\ w}} \left(\left(\sqrt{A_{v,w}}\right)^* \sqrt{\frac{\delta_w}{\delta_v}} |v\rangle\langle w| + \sqrt{A_{v,w}} \sqrt{\frac{\delta_v}{\delta_w}} |w\rangle\langle v| \right) \quad . \tag{B.1}$$

Then Δ has all row norms one, and $A = \|A'\| \cdot \Delta^\dagger \circ \Delta$.

Proof. Since A' has nonnegative entries, the principal eigenvector $|\delta\rangle$ is also nonnegative. Since $\tilde{G}_P(0^N)$ is a connected graph, $\delta_v > 0$ for every v. Hence Δ is well defined up to choice of sign of the square root, which doesn't matter.

By construction, for all v and w, $\Delta^*_{v,w}\Delta_{w,v} = A_{v,w}/\|A'\|$, i.e., $\|A'\| \cdot \Delta^\dagger \circ \Delta = A$. Furthermore, the squared norm of the v-th row of Δ is $\sum_w |\Delta_{v,w}|^2 = \frac{1}{\|A'\|}\frac{1}{\delta_v}\sum_w A'_{v,w}\delta_w = \frac{1}{\|A'\|}\frac{(A'\delta)_v}{\delta_v} = 1$. □

Remark B.3. *In defining Δ, we have evenly divided the complex phases of entries of A between red-black and black-red terms. However, any division of the phases would have worked. For example, Claim B.2 would also hold with Eq. (B.1) replaced by $\Delta = \frac{1}{\sqrt{\|A'\|}} \sum_{\substack{black\ v \\ red\ w}} \left(\frac{A^*_{v,w}}{\sqrt{|A_{v,w}|}} \sqrt{\frac{\delta_w}{\delta_v}} |v\rangle\langle w| + \sqrt{|A_{v,w}|\frac{\delta_v}{\delta_w}} |w\rangle\langle v| \right)$.*

We can now apply Szegedy's correspondence theorem [Sze04] to relate the spectrum of $\tilde{A}_{G_P}$ to that of a discrete-time coined quantum walk unitary.

Theorem B.4 ([Sze04])**.** *Let $\{|v\rangle : v \in V\}$ be an orthonormal basis for $\mathcal{H}_V$. For each $v \in V$, let $|\tilde{v}\rangle = |v\rangle \otimes \sum_{w \in V} \delta_{vw}|w\rangle \in \mathcal{H}_V \otimes \mathcal{H}_V$, where $\langle\tilde{v}|\tilde{v}\rangle = \sum_w |\delta_{vw}|^2 = 1$. Let $T = \sum_v |\tilde{v}\rangle\langle v|$ and $\Pi = TT^\dagger = \sum_v |\tilde{v}\rangle\langle\tilde{v}|$ be the projection onto the span of the $|\tilde{v}\rangle$s. Let $S = \sum_{v,w} |v,w\rangle\langle w,v|$, a swap. Let $U = (2\Pi - 1)S$, a swap followed by reflection about the span of the $|\tilde{v}\rangle$s. Let $M = T^\dagger S T = \sum_{v,w} \delta^*_{vw}\delta_{wv}|v\rangle\langle w|$.*

Then the spectral decomposition of U corresponds to that of M as follows: Take $\{|\lambda_\alpha\rangle\}$ a complete set of orthonormal eigenvectors of the Hermitian matrix M with respective eigenvalues λ_α. Let $R_\alpha = \mathrm{Span}\{T|\lambda_\alpha\rangle, ST|\lambda_\alpha\rangle\}$. Then $R_a \perp R_{\alpha'}$ for $\alpha \neq \alpha'$; let $R = \oplus_\alpha R_\alpha$. U fixes the spaces R_a and is $-S$ on $R^\perp$. The eigenvalues and eigenvectors of U within R_a are given by $\beta_{\alpha,\pm} = \lambda_\alpha \pm i\sqrt{1-\lambda_\alpha^2}$ and $(1 + \beta_{\alpha,\pm}S)T|\lambda_\alpha\rangle$, respectively.

A proof of Theorem B.4 in the above form is given in [CRŠZ07], and see [MNRS07].

Remark B.5 (Coined quantum walks)**.** *The operator $U = (2\Pi - 1)S$ in Theorem B.4 is known as a "coined quantum walk." S is known as the "step operator," and the reflection $(2\Pi - 1)$ is the "coin-flip operator." On the space R, $(2\Pi - 1)$ decomposes as $\sum_v |v\rangle\langle v| \otimes (\text{reflection about } \sum_w \delta_{vw}|w\rangle)$.*

In a classical random walk on a graph, a coin is flipped between each step to decide which adjacent vertex to step to next. In a coined quantum walk, on the other hand, the coin is maintained as part of the coherent quantum state, and is reflected between steps (also known as "diffusion").

Remark B.6. *Theorem B.4 can be viewed as giving a correspondence between coined quantum walks and classical random walks; in the special case that each $\delta_{vw} = \delta_{wv} \geq 0$, M is a classical random walk transition matrix. For general δ_{vw}, Theorem B.4 can be viewed a correspondence between coined quantum walks and continuous-time quantum walks. We use the theorem in the latter sense.*

Lemma B.7. *For Δ defined by Eq. (B.1) and with $\delta_{vw} = \Delta_{v,w}$, let U_{0^N} be the coined quantum walk operator $U_{0^N} = iU = i(2\Pi - 1)S$ in the notation of Theorem B.4. U_{0^N} acts on $\mathcal{H}_V \otimes \mathcal{H}_V$, where V is the vertex set of G_P. For $x \in \{0,1\}^N$, let $U_x = \tilde{O}_x U_{0^N}$, where $\tilde{O}_x$ applies a phase $(-1)^{x_i}$ to input vertex b_i and otherwise does nothing (see Eq. (4.9)). Then,*

- *If $\varphi(x) = 1$, there exist eigenvalue 1 and eigenvalue -1 normalized eigenstates of U_x each with $\Omega(1)$ support on $|a_O, b_O\rangle$.*
- *If $\varphi(x) = 0$, then U_x does not have any eigenstates supported on $|a_O, b_O\rangle$ with eigenvalues $\pm e^{i\lambda}$ for $|\lambda| \leq \arcsin\left(\frac{1}{\|A'\|} \frac{\epsilon}{\mathrm{Adv}(\varphi)}\right) = \Omega(1/\mathrm{Adv}(\varphi))$, where $\epsilon > 0$ is the constant of Theorem 4.16.*

Proof. Note that for an input vertex b_i on a span program input edge (a_i, b_i), the b_ith row of Δ is $\langle b_i|\Delta = \langle a_i|$. Define $\Delta(x)$ as follows: If $x_i = 1$, then let $\langle b_i|\Delta(x) = \langle b_i|$ (i.e., in the classical walk formulation, make b_i a probability sink), and let the other rows of $\Delta(x)$ be the same as those of $\Delta = \Delta(0^N)$.

In the notation of Theorem B.4 with each δ_{vw} set to the (v, w) entry of $\Delta(x)$, the vectors $|\tilde{v}\rangle$ do not depend on x if $v \notin \{b_i\}_{i \in I}$, whereas

$$|\tilde{b}_i\rangle = \begin{cases} |b_i, a_i\rangle & \text{if } x_i = 0 \\ |b_i, b_i\rangle & \text{if } x_i = 1 \end{cases}$$

Therefore, in $M = \Delta(x)^\dagger \circ \Delta(x)$, entries (a_i, b_i) and (b_i, a_i) are zeroed out when $x_i = 1$, while other entries are unchanged: so $M = \frac{1}{\|A'\|} A_{\tilde{G}_P(x)}$. Also, on R, $iU = i(2\Pi - 1)S$ is the same as U_x. So Theorem B.4 implies that the spectrum of $U_x = \tilde{O}_x U_{0^N}$ corresponds exactly to that of $A_{\tilde{G}_P(x)}/\|A'\|$. If the eigenvalues of $A_{\tilde{G}_P(x)}/\|A'\|$ are $\{\lambda_\alpha\}$, then the eigenvalues of U_x are given by $i\beta_{\alpha,\pm} = \pm\sqrt{1 - \lambda_\alpha^2} + i\lambda_\alpha$, i.e., $e^{i \arcsin \lambda_\alpha}$ and $-e^{-i \arcsin \lambda_\alpha}$.

In case $\varphi(x) = 1$, Theorem 4.16 promises that $A_{\tilde{G}_P(x)}$ has an eigenvalue-zero eigenstate with $\Omega(1)$ support on a_O. Denote this eigenstate by $|\lambda_\alpha = 0\rangle$. By Theorem B.4, $(1 \pm iS)T|\lambda_\alpha = 0\rangle$ are eigenstates of U_x with eigenvalues ± 1. Since $T|a_O\rangle = |a_O, b_O\rangle$, the eigenvectors $(1 \pm iS)T|\lambda_\alpha = 0\rangle$ each have $\Omega(1)$ support on $|a_O, b_O\rangle$. Moreover, this remains true even after renormalizing: T is an isometry, while the swap S is unitary, so $\|(1 \pm iS)T|\lambda_\alpha = 0\rangle\| \leq 2$.

The claim also follows for the case $\varphi(x) = 0$ by Theorems 4.16 and B.4. Every eigenstate of iU with support on $|a_O, b_O\rangle = T|a_O\rangle$ must be of the form $(1 + \beta_{\alpha,\pm} S)T|\lambda_\alpha\rangle = (1 + (\lambda_\alpha \pm i\sqrt{1 - \lambda_\alpha^2})S)T|\lambda_\alpha\rangle$. The terms which can overlap $T|a_O\rangle$ are either $\langle a_O|\lambda_\alpha\rangle$ (via T) or $\langle b_O|\lambda_\alpha\rangle$ (via ST). But by Theorem 4.16, both coefficients must be zero. Note that $\|A'\| = O(1)$ since the graph $\tilde{G}_P(0^N)$ has vertex degrees and edge weights all $O(1)$. Therefore, the spectral gap from zero of $A_{\tilde{G}_P(x)}/\|A'\|$ is only a constant factor worse than that of $A_{\tilde{G}_P(x)}$. □

B.2.2 Algorithm $\mathcal{ALG}$, correctness, and query and time complexity

Algorithm $\mathcal{ALG}$: Input $x \in \{0,1\}^N$, Output true/false.

1. Prepare an initial state on the output edge $|a_O, b_O\rangle$.
2. Run phase estimation on $U_x = \tilde{O}_x U_{0^N}$, with precision $\delta_p \leq \arcsin\left(\frac{1}{\|A'\|} \frac{\epsilon}{\mathrm{Adv}(\varphi)}\right)$ and small enough constant error rate δ_e.
3. Output true if the measured phase is 0 or π. Otherwise output false.

Correctness: Lemma B.7 implies that $\mathcal{ALG}$ is both complete and sound:

x	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
$x_1 \wedge \overline{x_3}$	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0
$x_1 \wedge x_2$	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 7: Function numbering scheme. We number each four-bit function by considering its sixteen-bit truth table as the binary representation of an integer. For example, $f(x_1, x_2, x_3, x_4) = x_1 \wedge \overline{x_3}$ is numbered $204 = 4 + 8 + 64 + 128$, while $x_1 \wedge x_2$ is numbered $15 = 1 + 2 + 4 + 8$. These two functions are equivalent, so in the table below we only list number 15, the smaller of the two.

- If $\varphi(x) = 1$, then $U(x)$ has eigenvalue-(± 1) eigenstates each with $\Omega(1)$ squared support on $|a_O, b_O\rangle$. The completeness parameter is at least this squared support minus the phase estimation error rate δ_e. For small enough constant δ_e, the completeness is $\Omega(1)$.
- If $\varphi(x) = 0$, then since the precision parameter δ_p is smaller than the promised gap away from ± 1 in Lemma B.7, phase estimation will output 0 or π only if there is an error. By choosing the error rate δ_e a small enough constant, the soundness error δ_e will be bounded away from the completeness parameter.

Therefore, algorithm $\mathcal{ALG}$ is correct. The constant gap between its completeness and soundness parameters can be amplified as usual.

Query and time complexity: Phase estimation of U_x with precision δ_p and error rate δ_e requires $O(1/(\delta_p \delta_e))$ calls to $U_x = \tilde{O}_x U_{0^N}$ [CEMM98]. Therefore, $\mathcal{ALG}$ makes $O(\mathrm{Adv}(\varphi))$ queries to the input oracle O_x.

The time-efficiency claim of Theorem 4.7 is slightly more complicated. Here, we need to allow a preprocessing phase in which the algorithm can compute $A_{\tilde{G}_P(0^N)}$ and in particular (approximations to) the coin diffusion operators in U_{0^N}. This preprocessing depending on φ, but not x, takes $\mathrm{poly}(N)$ time. The algorithm then needs coherent access to the precomputed information in order to apply efficiently the coin diffusion operators. For further details, see [CRŠZ07].

This completes the proof of Theorem 4.7. □

C Table of functions on up to four bits

Which four-bit gates can be added to the gate set $\mathcal{S}$ without affecting the correctness of Theorem 4.8 or Theorem 4.7? As summarized in Section 5.1, we have made partial progress toward answering this question. In this appendix, we present a table that lists all the four-bit gates, up to equivalences, and says what we know for each gate.

Definition C.1. *Two k-bit boolean functions $f_1, f_2 : \{0,1\}^k \to \{0,1\}$ are equivalent if there exists a string $y \in \{0,1\}^k$ and a k-element permutation $\sigma \in S_k$ such that either for all $x \in \{0,1\}^k$ $f_1(x) = f_2(\sigma(x) \oplus y)$, or for all $x \in \{0,1\}^k$ $f_1(x) = \neg f_2(\sigma(x) \oplus y)$. Here for a bit string $x = x_1 x_2 \ldots x_k \in \{0,1\}^k$, $\sigma(x) \in \{0,1\}^k$ is defined as $x_{\sigma^{-1}(1)} x_{\sigma^{-1}(2)} \ldots x_{\sigma^{-1}(k)}$.*

That is, two functions are equivalent if they differ by permuting the inputs, and complementing a subset of the input bits and output bit. There are 222 inequivalent four-bit functions. We number each function for reference by considering its sixteen-bit truth table as the binary representation of an integer as in Figure 7.

The table begins with the fourteen inequivalent functions of Figure 5 that depend on at most three input bits. The 208 functions that depend on all four input bits are first sorted according to their polynomial degree [Amb06a] and then by their non-negative-weight and general adversary bounds, Adv and $\mathrm{Adv}^{\pm}$. If these two bounds are equal, then we state only the first of the two. The numerical adversary bounds were taken from [HLŠ06].

For each function, we give its {AND, OR, NOT} formula size. Note that by Lemma 4.12, a function of formula size k has a trivial span program with witness size $\sqrt{k}$. The table includes comments on some of the functions. These comments might include, for example, the smallest-size {AND, OR, NOT} formula for an equivalent function. For some of the functions, e.g., #7128, we have listed in the Status column the witness size for the best span program we have found that computes the function. Those functions for which we know an optimal span program, with witness size matching the adversary bound, are marked with a checkmark (✓), together with a brief justification. For example, a reference to Lemma 4.12 means that the optimal span program follows from composing two simpler span programs based on an OR or PARITY gate. The lower bound for such functions follows from a simple fact on composition of adversary lower bounds:

Lemma C.2. *For two functions f_1, f_2 on disjoint inputs,* $\mathrm{Adv}(f_1 \oplus f_2) = \mathrm{Adv}(f_1) + \mathrm{Adv}(f_2)$*, and* $\mathrm{Adv}(f_1 \vee f_2) = \sqrt{\mathrm{Adv}(f_1)^2 + \mathrm{Adv}(f_2)^2}$.

For a function f marked "opt. NAND," the optimal span program comes from optimizing the edge weights of the tree corresponding to a minimal-size NAND formula computing f.

Listed below the table are the optimal adversary matrices (Definition 4.3) and optimal span programs for all the four-bit functions for which we know an optimal span program, except when one bound or the other follows from other stated results, e.g., from Lemma C.2 or Lemma 4.12. In a Mathematica file included as an electronic supplement to this article, we have included code to verify the adversary bound and span program witness size calculations.

For details on any of the span programs referenced with witness size that does not match the adversary lower bound, e.g., for function #7128, please contact one of the authors.

Functions depending on up to three input bits:

#	Size	Adv	$\mathrm{Adv}^{\pm}$	Status	Comments
0	0	0	–	✓ trivial	constant 0
255	1	1	–	✓ trivial	x_1
15	2	$\sqrt{2}$	–	✓ Lemma 4.12	$x_1 \vee x_2$
3	3	$\sqrt{3}$	–	✓ Lemma 4.12	$x_1 \vee x_2 \vee x_3$
63	3	$\sqrt{3}$	–	✓ Lemma 4.12	$x_1 \vee (x_2 \wedge x_3)$
4080	4	2	–	✓ Lemma 4.12	$x_1 \oplus x_2$
975	4	2	–	✓ Lemma 4.12	$(x_3 \wedge x_2) \vee (\overline{x_3} \wedge x_1)$
831	5	2	–	✓ Claim 4.9	MAJ_3
960	6	$3/\sqrt{2}$	–	✓ Claim 4.10	EQUAL_3
963	5	$\sqrt{3+\sqrt{3}}$	–	✓ Claim 4.11	$(x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2})$
60	5	$\sqrt{5}$	–	✓ Lemma 4.12	$x_1 \vee (x_2 \oplus x_3)$
1020	6	$1+\sqrt{2}$	–	✓ Lemma 4.12	$x_1 \oplus (x_2 \wedge x_3)$
828	8	$\sqrt{7}$	–	✓ Lemma 4.12	$\mathrm{EXACT}_{2 \text{ of } 3}$
15555	10	3	–	✓ Lemma 4.12	$x_1 \oplus x_2 \oplus x_3$

Functions depending on four input bits, with polynomial degree either two or three:

#	Size	Adv	$\mathrm{Adv}^{\pm}$	Status	Comments
7128	10	2.50000	2.51353	2.77394	sorted input bits [Amb06a], $(x_1 \wedge ((x_2 \wedge x_3) \vee (\overline{x_3} \wedge \overline{x_4}))) \vee (\overline{x_1} \wedge ((\overline{x_2} \wedge \overline{x_3}) \vee (x_3 \wedge x_4)))$
863	5	2.00000	2.07136	2.22833	monotone two adjacent 1s, $(x_1 \wedge x_2) \vee (x_4 \wedge (x_1 \vee x_3))$
427	5	2.18398	2.20814	2.22833	$\#975(x_1 \wedge x_2, x_3, x_4)$

#	Size	Adv	Adv$^{\pm}$	Status	Comments
27	5	$\sqrt{5}$	–	✓ Lemma 4.12	$x_1 \wedge \#975(x_2, x_3, x_4)$
393	6	$4/\sqrt{3}$	–	✓ opt. NAND	$(x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2} \wedge x_4)$
383	7	2.30278	2.34406	$\sqrt{4+\sqrt{3}}$	$(x_1 \wedge x_2 \wedge x_3) \vee ((x_1 \vee x_2 \vee x_3) \wedge x_4)$
126	7	$\sqrt{11/2}$	–	✓ Lemma 4.12	$x_1 \wedge \neg\,\mathrm{EQUAL}_3(x_2, x_3, x_4)$
24	7	$\sqrt{11/2}$	–	✓ Lemma 4.12	$x_1 \wedge \mathrm{EQUAL}_3(x_2, x_3, x_4)$
303	6	2.35829	–	$1+\sqrt{2}$	$((x_1 \vee x_2) \wedge x_3) \vee ((\overline{x_1} \vee x_2) \wedge x_4)$, span program size 5
495	6	$1+\sqrt{2}$	–	✓ opt. NAND	$\#975(x_1, x_2, x_3 \wedge x_4)$
989	6	$1+\sqrt{2}$	–	✓ opt. gadget	$(x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge (\overline{x_2} \vee x_4))$
965	7	2.41531	2.42653	2.59234	$(x_1 \wedge x_2 \wedge x_3) \vee (x_4 \wedge \overline{x_3}) \vee (\overline{x_4} \wedge \overline{x_2})$
987	7	2.43128	2.44711		
424	8	$\sqrt{6}$	–	✓ opt. NAND	$\mathrm{EQUAL}_3(x_1, x_2, x_3 \wedge x_4)$
2034	7	2.47323	2.48653		
429	7	2.48837	2.50826		
490	8	2.52207	2.52326		
281	8	2.51464	2.54019		
2022	8	2.55719	2.58189		
1968	8	$2\sqrt{5/3}$	–	✓ opt. NAND	$(x_1 \wedge x_2 \wedge (x_3 \vee x_4)) \vee (\overline{x_1} \wedge \overline{x_2} \wedge (\overline{x_3} \vee x_4))$
1782	7	2.56155	2.59040	2.61804	$\#975(x_1 \oplus x_2, x_3, x_4)$
984	8	2.58539	2.60282		
1973	8	2.61704	2.63510		
1910	8	$\sqrt{7}$	–	✓ opt. NAND	$(x_1 \wedge x_2 \wedge x_3 \wedge x_4) \vee (\overline{x_1} \wedge \overline{x_2}) \vee (\overline{x_3} \wedge \overline{x_4})$
317	8	$\sqrt{7}$	–	✓ Lemma 4.12	$(x_1 \wedge x_2 \wedge x_3) \vee \mathrm{MAJ}_3(\overline{x_2}, \overline{x_3}, x_4)$
858	8	2.64575	2.64658	$\sqrt{5+2\sqrt{2}}$	$(x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_3} \wedge \overline{x_4}) \vee (\overline{x_1} \wedge (\overline{x_2} \vee \overline{x_3}))$
300	9	2.69932	2.70595		
6030	10	2.70928	–	3	$(x_1 \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee x_4)) \vee (\overline{x_1} \wedge (\overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_4}))$, classical three-query alg.
894	10	2.70808	2.71982		
1714	10	2.75018	2.75944		
1980	9	2.75779	2.77469		
1719	9	2.76916	2.78319		
366	10	2.76569	2.80341		
6042	10	2.84104	2.84923		
1716	10	2.86854	2.88186		
1680	12	3	–	✓ opt. NAND	$\mathrm{EQUAL}_3(x_1, x_2, x_3 \oplus x_4)$
1695	10	3	–	✓ opt. NAND	$\#975(x_1, x_2, x_3 \oplus x_4)$, tight classical alg.
5790	10	3	–	✓ opt. NAND	$(x_1 \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee x_4)) \vee (\overline{x_1} \wedge (x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_3))$, tight classical alg.
7140	10	3	–	✓ Lemma 4.12	$x_1 \oplus \#975(x_2, x_3, x_4)$, tight classical alg.
1683	11	3.00283	3.01009		
5766	12	3.02533	3.03595		
6627	12	3.01470	3.04933		
5783	12	3.03917	3.07294		
6375	14	$1+\sqrt{9/2}$	–	✓ Lemma 4.12	$x_1 \oplus \mathrm{EQUAL}_3(x_2, x_3, x_4)$

Functions depending on four input bits, with polynomial degree four:

#	Size	Adv	Adv$^{\pm}$	Status	Comments
1	4	2	–	✓ Lemma 4.12	$x_1 \wedge x_2 \wedge x_3 \wedge x_4$

#	Size	Adv	$\text{Adv}^{\pm}$	Status	Comments
127	4	2	–	✓ Lemma 4.12	$x_1 \vee (x_2 \wedge x_3 \wedge x_4)$
31	4	2	–	✓ Lemma 4.12	$x_1 \wedge (x_2 \vee (x_3 \wedge x_4))$
7	4	2	–	✓ Lemma 4.12	$x_1 \wedge x_2 \wedge (x_3 \vee x_4)$
855	4	2	–	✓ Lemma 4.12	$(x_1 \vee x_3) \wedge (x_2 \vee x_4)$
319	6	2.17533	2.20453	2.27731	$(x_1 \wedge (x_2 \vee x_3)) \vee (x_2 \wedge x_3 \wedge x_4)$
431	5	2.20635	2.20721	2.22835	$((x_1 \vee x_2) \wedge x_3) \vee (\overline{x_1} \wedge x_4)$
23	6	$\sqrt{5}$	–	✓ Lemma 4.12	$x_1 \wedge \text{MAJ}_3(x_2, x_3, x_4)$
399	6	2.22595	2.25274		
287	6	$\sqrt{3+\sqrt{5}}$	–	✓ Lemma 5.2	$\text{MAJ}_3(x_1, x_2, x_3 \wedge x_4)$
384	8	$4/\sqrt{3}$	–	✓ Claim 4.10	EQUAL_4
447	7	2.30278	2.31707		
385	7	$(1/2)\sqrt{13+\sqrt{73}}$	–	✓ opt. NAND	$(x_1 \vee (x_2 \wedge x_3)) \wedge (\overline{x_1} \vee (\overline{x_2} \wedge \overline{x_3} \wedge x_4))$
395	6	2.29062	2.32499		
2032	6	$\sqrt{(7+\sqrt{17})/2}$	–	✓ opt. NAND	$\#963(x_1, x_2, x_3 \vee x_4)$
426	6	$\sqrt{(7+\sqrt{17})/2}$	–	✓ opt. NAND	$\#963(x_1, x_2, x_3 \wedge x_4)$
967	6	2.36698	2.37004		
25	6	$\sqrt{4+\sqrt{3}}$	–	✓ Lemma 4.12	$x_1 \wedge \#963(x_2, x_3, x_4)$
61	6	$\sqrt{4+\sqrt{3}}$	–	✓ Lemma 4.12	$x_1 \vee \#963(x_2, x_3, x_4)$
981	7	2.39489	2.40490		
283	7	2.40091	2.42364		
983	6	2.42266	2.42424		
409	7	2.39120	2.42693		
961	7	2.43531	2.43869		
111	6	$\sqrt{6}$	–	✓ Lemma 4.12	$x_1 \vee \#60(x_2, x_3, x_4)$
1638	6	$\sqrt{6}$	–	✓ Lemma 4.12	$(x_1 \wedge x_2) \vee (x_3 \oplus x_4)$
279	8	$\sqrt{6}$	–	✓ Example 5.1	$\text{Threshold}_{2 \text{ of } 4}$
6	6	$\sqrt{6}$	–	✓ Lemma 4.12	$x_1 \wedge x_2 \wedge (x_3 \oplus x_4)$
387	7	2.48662	2.50251		
859	7	2.49857	2.50575		
988	7	2.53791	2.53835		
411	8	2.52192	2.54009		
1654	7	2.51758	2.54347		
445	9	2.53019	2.55017		
425	7	2.55654	–	2.55654	$\#963(x_1 \wedge x_2, x_3, x_4)$, exact exp. for Adv?
494	7	2.55654	–	2.55654	$\#963(x_1 \vee x_2, x_3, x_4)$, exact exp. for Adv?
2018	8	2.54502	2.55711		
2016	8	2.55128	2.55874		
2033	8	2.56155	–	2.59163	$((x_1 \vee x_2) \wedge x_3 \wedge x_4) \vee (x_2 \wedge x_3) \vee (\overline{x_3} \wedge \overline{x_4})$
491	7	2.56388	2.57901	2.64302	$(x_1 \wedge x_2 \wedge x_3) \vee (x_4 \wedge (\overline{x_1} \vee (\overline{x_2} \wedge \overline{x_3})))$
879	8	2.57641	2.58289		
415	8	2.57096	2.58436		
428	8	2.59386	2.60618		
430	7	2.60716	2.60975		
30	7	$\sqrt{4+2\sqrt{2}}$	–	✓ Lemma 4.12	$x_1 \wedge \#1020(x_2, x_3, x_4)$
2019	8	2.61439	2.62037		
488	10	2.62705	–		
1639	8	$\sqrt{7}$	–	✓ opt. gadget	$(x_1 \wedge x_2) \vee \text{MAJ}_3(\overline{x_1} \wedge \overline{x_2}, x_3, x_4)$
1969	9	2.64898	2.65285		

#	Size	Adv	Adv$^{\pm}$	Status	Comments
1778	8	2.66716	2.66873		
985	8	2.65949	2.67406		
980	9	2.67345	2.67735		
1650	8	2.67869	2.68369		
391	9	2.68828	2.70027		
878	9	2.68845	2.70057		
1918	10	2.70131	–		$(\overline{x_1} \wedge x_2 \wedge (x_3 \vee x_4)) \vee (x_1 \wedge \neg\,\mathrm{EQUAL}_4(x_2, x_3, x_4))$
386	9	2.67082	2.70300		
966	8	2.70246	2.70500		
367	8	2.70387	2.70585		
1662	9	2.69544	2.70815		
1776	8	$\sqrt{(9+\sqrt{33})/2}$	–	✓ opt. NAND	$\#963(x_1, x_2, x_3 \oplus x_4)$
280	10	2.71411	2.71639		
408	9	2.69951	2.71811		
301	8	2.71328	2.71856		
1647	8	$1+\sqrt{3}$	–	✓ Lemma 5.2	$\mathrm{MAJ}_3(x_1, x_2, x_3 \oplus x_4)$
2040	8	$1+\sqrt{3}$	–	✓ Lemma 4.12	$x_1 \oplus (x_2 \wedge (x_3 \vee x_4))$
510	8	$1+\sqrt{3}$	–	✓ Lemma 4.12	$x_1 \oplus (x_2 \wedge x_3 \wedge x_4)$
1634	8	2.74013	2.74148		
856	9	2.73510	2.74171		
892	9	2.72608	2.74205		
316	9	2.74228	2.74790		
862	9	2.74628	2.75157		
829	9	2.75490	2.76384		
990	8	2.76066	2.76706		
1651	8	2.75952	2.76736		
1715	9	2.76490	2.77389		
489	9	2.78575	–	2.86182	$(x_1 \wedge x_2 \wedge x_3) \vee (x_4 \wedge (\overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee (\overline{x_2} \wedge \overline{x_3})))$, x_2 and x_3 symmetrical
6040	10	2.77499	2.79246		
1914	9	2.79485	–	2.85539	$(x_1 \wedge x_2 \wedge (x_3 \vee x_4)) \vee (x_3 \wedge x_4) \vee (\overline{x_1} \wedge \overline{x_3} \wedge \overline{x_4})$, x_3 and x_4 symmetrical
282	9	2.80369	–	2.92535	$(\overline{x_1} \wedge \overline{x_3} \wedge \overline{x_4}) \vee (x_3 \wedge x_4) \vee ((x_1 \vee x_2) \wedge (x_3 \vee x_4))$, x_2 and x_3 symmetrical
1972	9	2.79678	2.80499		
893	9	2.79694	2.80607		
444	10	2.80787	2.81297		
874	10	2.81477	2.81815		
107	9	$2\sqrt{2}$	–	✓ Lemma 4.12	$x_1 \wedge \#828(x_2, x_3, x_4)$
1632	8	$2\sqrt{2}$	–	✓ Lemma 4.12	$(x_1 \oplus x_2) \wedge (x_3 \oplus x_4)$
22	9	$2\sqrt{2}$	–	✓ Lemma 4.12	$x_1 \wedge \#828(x_2, x_3, x_4)$
6014	12	$2\sqrt{2}$	–	✓ opt. gadget	$\mathrm{EXACT}_{2 \text{ or } 3 \text{ of } 4} = \mathrm{Thr}_{\cdot 2 \text{ of } 4}(x_1, x_2, x_3, x_4) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$
854	8	$2\sqrt{2}$	–	✓ Lemma 4.12	$(x_1 \wedge x_2) \oplus (x_3 \wedge x_4)$
6060	10	2.82034	2.83150		
875	9	2.83428	2.83570		
2017	10	2.83417	2.83655		
1712	10	2.84346	2.84354		

#	Size	Adv	Adv$^{\pm}$	Status	Comments
857	9	2.82909	2.85105		
1718	9	2.84413	2.85900		
382	11	2.87999	–		
362	11	2.88004	2.88205		
5758	11	2.89586	–		
876	10	2.89815	2.90403		
318	10	2.90163	2.90404		
407	11	2.89638	2.90417		
410	10	2.90592	2.91505		
446	10	2.91254	2.91560		
1974	10	2.91560	2.91835		
363	10	2.92040	2.92652		
1658	10	2.92940	2.93141		
5774	11	2.92267	2.93717		
1717	10	2.93360	2.94668		
1777	10	2.96176	–		
982	10	2.96425	2.96696		
1635	10	2.98641	2.98673		
6120	12	3	–	✓ Lemma 4.12	$x_1 \oplus \mathrm{MAJ}_3(x_2, x_3, x_4)$
1713	11	2.99622	3.00474		
1912	10	$\sqrt{5+\sqrt{17}}$	–	✓ opt. NAND	$\#828(x_1 \vee x_2, x_3, x_4)$
286	10	$\sqrt{5+\sqrt{17}}$	–	✓ opt. NAND	$\#828(x_1 \wedge x_2, x_3, x_4)$
5786	11	3.01265	3.02051		
1687	11	3.01018	3.02207		
1681	12	3.02473	3.02629		
360	12	3.04017	3.04042		
1659	11	3.04139	3.04288		
6625	12	3.02185	3.04627		
5820	11	3.03542	3.04710		
1725	12	3.04048	3.05100		
877	11	3.04498	3.05270		
390	12	3.03755	3.05354		
872	12	3.06480	3.06823		
5782	11	3.06111	3.07244		
5784	12	3.07314	3.07400		
5742	12	3.09004	3.09058		
1686	11	3.11060	–		$\#963(x_1 \oplus x_2, x_3, x_4)$
5804	12	3.11134	3.11717		
2025	12	3.12714	3.12849		
6038	12	3.12842	3.12999		
414	12	3.12805	3.13416		
5767	13	3.13610	3.13705		
361	11	3.14864	–		
5787	12	3.14761	3.15425		
105	11	$\sqrt{10}$	–	✓ Lemma 4.12	$x_1 \wedge (x_2 \oplus x_3 \oplus x_4)$
278	12	$\sqrt{10}$	–	✓ Lemma 4.12	$\mathrm{EXACT}_{1 \text{ of } 4} = \mathrm{Thr}_{3 \text{ of } 4}(\overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3 \vee x_4)$
1656	12	3.16284	3.16420		
6630	12	$1+\sqrt{3+\sqrt{3}}$	–	✓ Lemma 4.12	$x_1 \oplus \#963(x_2, x_3, x_4)$
1721	12	3.19509	3.19570		

#	Size	Adv	$\text{Adv}^{\pm}$	Status	Comments
1913	12	3.19640	–		
5763	13	3.21393	3.21633		
5771	13	3.21305	3.21888		
1643	12	3.21930	–		
1633	12	3.23607	–	3.33513	$((x_1 \oplus x_2) \wedge (x_3 \oplus x_4)) \vee (x_1 \wedge x_2 \wedge x_3 \wedge x_4)$
1785	12	$1+\sqrt{5}$	–	✓ Lemma 4.12	$x_1 \oplus \#60(x_2, x_3, x_4)$
873	12	3.26876	3.27185		
5785	12	3.27183	3.27189		
5738	14	3.28207	–		
5805	14	3.34542	3.34781		
5761	15	3.36028	–		$\text{EXACT}_{1 \text{ or } 4 \text{ of } 4}$
406	14	3.36637	3.37384		
1657	14	3.39009	3.39051		
5769	14	3.39400	–		
7905	12	$2+\sqrt{2}$	–	✓ Lemma 4.12	$x_1 \oplus x_2 \oplus (x_3 \wedge x_4)$
5736	16	$2\sqrt{3}$	–	✓ Lemma 4.12	$\text{EXACT}_{2 \text{ of } 4} = \text{Thr}._{2 \text{ of } 4}(x_1, x_2, x_3, x_4) \wedge \text{Thr}._{2 \text{ of } 4}(\overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4})$
5801	14	3.51041	3.51129		
5739	15	3.51414	–		
1641	14	$\sqrt{7+\sqrt{33}}$	–	✓ opt. NAND	$\#828(x_1 \oplus x_2, x_3, x_4)$
5865	16	$1+\sqrt{7}$	–	✓ Lemma 4.12	$x_1 \oplus \#828(x_2, x_3, x_4)$
5737	16	3.78478	–		$\text{EXACT}_{2 \text{ or } 4 \text{ of } 4}$
27030	16	4	–	✓ Lemma 4.12	PARITY_4, $x_1 \oplus x_2 \oplus x_3 \oplus x_4$

We now list optimal adversary matrices and optimal span programs for the four-bit functions for which we know an optimal span program. For the adversary matrices, we include only the rows and columns with nonzero entries.

Several of the four-bit functions, for example #287, $\text{MAJ}_3(x_1, x_2, x_3 \wedge x_4)$, or #1647, $\text{MAJ}_3(x_1, x_2, x_3 \oplus x_4)$, are compositions of a three-bit function with a two-bit function. Therefore, to consolidate cases and increase the generality of our results, it will sometimes be convenient to give span programs for three-bit functions with unbalanced input complexities. Lemma 5.2, for example, implies optimal span programs for both functions #287 and #1647. For the adversary lower bound, it is also convenient to define the general adversary bound with costs [HLŠ07, HLŠ05]:

Definition C.3 (Adversary bound with costs)**.** *Let $f : \{0,1\}^k \to \{0,1\}$, and let $\alpha \in \mathbf{R}_+^n$ be a vector of positive reals. Define*

$$\text{Adv}_\alpha^{\pm}(f) = \max_{\Gamma \neq 0} \frac{\|\Gamma\|}{\max_i \frac{1}{\alpha_i} \|\Gamma \circ D_i\|} \ , \tag{C.1}$$

where the matrices D_i are defined as in Definition 4.3. The maximum is over all $2^k \times 2^k$ nonzero, symmetric matrices Γ satisfying $\langle x|\Gamma|y\rangle = 0$ if $f(x) = f(y)$.

This weighted version of the adversary bound composes nicely, as shown by the following generalization of Theorem 4.6, still a special case of [HLŠ07, Theorem 13]:

Theorem C.4 ([HLŠ07])**.** *Let $f = g \circ (h_1, \ldots, h_k)$ and let $\alpha = (\text{Adv}^{\pm}(h_1), \ldots, \text{Adv}^{\pm}(h_k))$. Then $\text{Adv}^{\pm}(f) \geq \text{Adv}_\alpha^{\pm}(f)$.*

[HLŠ07, Theorem 12] shows that the weighted version of the *nonnegative* adversary bound composes exactly, i.e., the $\geq$ sign can be replaced with equality, but equality is not known to hold for the general adversary bound.

Theorem C.4 lets us compute the adversary bound for the four-bit function #287, $\mathrm{MAJ}_3(x_1, x_2, x_3 \wedge x_4)$, for example, by considering $2^3 \times 2^3$ adversary matrices for the three-bit majority function with costs $\alpha = (1, 1, \sqrt{2})$.

C.1 Function #831, MAJ_3, with partly unbalanced inputs:

If the first two inputs have equal costs 1 and the third input has cost β, then the optimal adversary matrix for function #831, MAJ_3, comes from

$$\Gamma = \begin{array}{c} \\ 100 \\ 010 \\ 001 \end{array} \begin{array}{c} \begin{array}{ccc} 011 & 101 & 110 \end{array} \\ \begin{pmatrix} 0 & \beta & 1 \\ \beta & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$

The adversary bound is $\mathrm{Adv}^{\pm}_{(1,1,\beta)}(\mathrm{MAJ}_3) = \frac{1}{2}\left(\sqrt{8+\beta^2} + \beta\right)$. An optimal span program for this function with equal first two input complexities is given in Lemma 5.2. Note that this also settles the complexities of functions #287 and #1647.

C.2 Function #975 with partly unbalanced inputs:

Function #975 is $(x_3 \wedge x_2) \vee (\overline{x_3} \wedge x_1)$. If the first two inputs have equal costs 1 and the third input has cost β, then the optimal adversary matrix comes from

$$\Gamma = \begin{array}{c} \\ 100 \\ 011 \end{array} \begin{array}{c} \begin{array}{cc} 010 & 101 \end{array} \\ \begin{pmatrix} 1 & \beta \\ \beta & 1 \end{pmatrix} \end{array}$$

The adversary bound is $\mathrm{Adv}^{\pm}_{(1,1,\beta)}(\#975) = \beta + 1$. A span program with matching witness size is

$$X_J = (\{x_1\}\ \{x_3\}\ \{\overline{x_3}\}\ \{x_2\})$$
$$t = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad v_J = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

This function can be evaluated as an "if-then-else" statement, by evaluating the third input x_3 at cost β and then one of x_1 or x_2 at cost 1. A span program with witness size $\beta + 1$ is thus not surprising. Note that this also settles the complexities of functions #495 and #1695.

C.3 Function #960, EQUAL_3, with partly unbalanced inputs:

For function #960, EQUAL_3, the adversary bound when the first two inputs have equal costs 1 and the third input has cost β is

$$\mathrm{Adv}^{\pm}_{(1,1,\beta)}(\mathrm{EQUAL}_3) = \begin{cases} \beta + \sqrt{2-\beta^2} & \text{if } 0 < \beta \leq \sqrt{2/5} \\ \sqrt{\frac{3}{2}(2+\beta^2)} & \text{if } \sqrt{2/5} \leq \beta \leq 2 \\ \beta + 1 & \text{if } \beta \geq 2 \end{cases}.$$

Indeed, optimal adversary matrices are

$$\begin{array}{c} \\ 001 \\ 110 \end{array} \begin{array}{c} \begin{array}{cc} 000 & 111 \end{array} \\ \begin{pmatrix} \beta & 1 \\ 1 & \beta \end{pmatrix} \end{array}, \qquad \begin{array}{c} \\ 100 \\ 010 \\ 001 \\ 011 \\ 101 \\ 110 \end{array} \begin{array}{c} \begin{array}{cc} 000 & 111 \end{array} \\ \begin{pmatrix} 2\alpha & \alpha \\ 2\alpha & \alpha \\ 2 & 1 \\ \alpha & 2\alpha \\ \alpha & 2\alpha \\ 1 & 2 \end{pmatrix} \end{array} \qquad \text{and} \qquad \begin{array}{c} \\ 100 \\ 010 \\ 011 \\ 101 \end{array} \begin{array}{c} \begin{array}{cc} 000 & 111 \end{array} \\ \begin{pmatrix} \sqrt{2-\beta^2} & \beta \\ \sqrt{2-\beta^2} & \beta \\ \beta & \sqrt{2-\beta^2} \\ \beta & \sqrt{2-\beta^2} \end{pmatrix} \end{array}$$

for the ranges $0 < \beta \leq \sqrt{2/5}$, $\sqrt{2/5} \leq \beta \leq 2$ and $\beta \geq 2$, respectively, where $\alpha = \sqrt{\frac{4-\beta^2}{5\beta^2-2}}$.

The optimal span program is

$$X_J = (\{x_3\}\ \{x_1, x_2\}\ \{\overline{x_1}, \overline{x_2}\}\ \{\overline{x_3}\})$$
$$t = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad v_J = \begin{pmatrix} 0 & 1 & 1 & 0 \\ w & 1 & 0 & 0 \\ 0 & 0 & 1 & w \end{pmatrix},$$

where $w = \left(\frac{\beta+\sqrt{2-\beta^2}}{2(1-\beta^2)}\right)^{1/2}$ for $\beta \leq \sqrt{2/5}$, $w = 1/\sqrt{\beta}$ for $\sqrt{2/5} \leq \beta \leq 2$, and $w = 1/\sqrt{2}$ for $\beta \geq 2$.

Note that this also settles the complexities of functions #424, $\mathrm{EQUAL}_3(x_1, x_2, x_3 \wedge x_4)$, and #1680, $\mathrm{EQUAL}_3(x_1, x_2, x_3 \oplus x_4)$.

C.4 Function #963 with partly unbalanced inputs:

See Claim 4.11 for the balanced complexity case for function #963, $(x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2})$. If the first two, symmetrical inputs have equal costs 1 and the third input has cost β, the the optimal adversary matrix comes from

$$\Gamma = \begin{array}{c} \\ 011 \\ 101 \\ 110 \end{array} \begin{array}{c} \begin{array}{ccc} 000 & 001 & 111 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & \beta \end{pmatrix} \end{array}$$

The adversary bound is $\mathrm{Adv}^{\pm}_{(1,1,\beta)}(\#963) = \frac{1}{\sqrt{2}}\left(5+\beta^2+\sqrt{9+2\beta^2+\beta^4}\right)^{1/2}$. A span program with matching witness size is

$$X_J = (\{x_1\}\ \{x_2, x_3\}\ \{\overline{x_2}, \overline{x_3}\})$$
$$t = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad v_J = \begin{pmatrix} 0 & w_1 & 1 \\ 1 & w_2 & 0 \end{pmatrix},$$

where $w_1 = \frac{1}{2}\left(1+\beta^2+\sqrt{9+2\beta^2+\beta^4}\right)^{1/2}$ and $w_2 = \frac{1}{\sqrt{2}\beta}\left(-3+\beta^2+\sqrt{9+2\beta^2+\beta^4}\right)^{1/2}$. Note that this also settles the complexities of functions #2032, #426 and #1776.

C.5 Function #828, $\mathrm{EXACT}_{2\,\mathrm{of}\,3}$, with partly unbalanced inputs:

Function #828 is $\mathrm{EXACT}_{2\ \mathrm{of}\ 3}(x_1, x_2, x_3) = \mathrm{MAJ}_3(x_1, x_2, x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$. If the first two inputs have equal costs 1 and the third input has cost β, then the optimal adversary matrix comes from

$$\Gamma = \begin{array}{c} \\ 001 \\ 010 \\ 100 \\ 111 \end{array} \begin{array}{c} \begin{array}{ccc} 110 & 101 & 011 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & \beta \\ 1 & \beta & 0 \\ \beta & 1 & 1 \end{pmatrix} \end{array}$$

The adversary bound is $\mathrm{Adv}^{\pm}_{(1,1,\beta)}(\#828) = \left(3+\beta^2+\sqrt{1+8\beta^2}\right)^{1/2}$. A span program with matching witness size is

$$X_J = (\{\overline{x_1}, \overline{x_2}\}\ \{\overline{x_3}\}\ \{x_1\}\ \{x_2\}\ \ \{x_3\}\ )$$
$$t = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad v_J = \begin{pmatrix} 1 & 0 & w_2 & w_2 & w_1/\sqrt{2} \\ 0 & 0 & i & -i & w_1 \\ w_1 & 1 & 0 & 0 & 0 \end{pmatrix},$$

where $w_1^2 = \left(\sqrt{8\beta^2+1}-1\right)/(2\beta)$ and $w_2^2 = 1/\left(\sqrt{8\beta^2+1}+3\right)$. Note that this also settles the complexities of functions #1912, #286 and #1641.

C.6 Function #393:

The optimal adversary matrix and a matching span program for function #393 are

$$\begin{array}{c} \\ 0001 \\ 1110 \\ 0011 \\ 1111 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} 0010 & 1101 & 0111 & 1011 \end{array} \\ \begin{pmatrix} 1 & 3 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 2\sqrt{2} & 0 & 3 & 3 \\ 0 & 2\sqrt{2} & 3 & 3 \end{pmatrix} \end{array} \qquad t = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \begin{array}{c} X_J = (\{x_1, x_2\} \;\; \{x_3\} \;\; \{\overline{x_1}, \overline{x_2}\} \;\; \{x_4\}\,) \\ v_J = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & \sqrt{3/2} & 0 & 0 \\ 0 & 0 & 1 & \sqrt{3/2} \end{pmatrix} \end{array}.$$

C.7 Function #989:

The optimal adversary matrix and a matching span program for function #989 are

$$\begin{array}{c} \\ 0011 \\ 0110 \\ 1101 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} 0010 & 0111 & 1100 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \end{array} \qquad t = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \begin{array}{c} X_J = (\{\overline{x_1}, x_2\} \; \emptyset \;\; \{\overline{x_2}\} \;\; \{\overline{x_3}\} \;\; \{x_4\} \;\; \{\overline{x_4}\}) \\ v_J = \begin{pmatrix} 0 & 1 & 1 & 1 & 2^{1/4} & 1 \\ 0 & 1 & 2^{-1/4} & 2^{-1/4} & 1 & 1 \\ 2^{1/4} & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

C.8 Function #1968:

The optimal adversary matrix and a matching span program for function #1968 are

$$\begin{array}{c} \\ 0010 \\ 1100 \\ 1001 \\ 0101 \\ 0111 \\ 1011 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} 0000 & 1110 & 0011 & 1101 \end{array} \\ \begin{pmatrix} 3\sqrt{7} & 2\sqrt{7} & 3\sqrt{7} & 0 \\ 2\sqrt{7} & 3\sqrt{7} & 0 & 3\sqrt{7} \\ 5 & 0 & 2 & 7 \\ 5 & 0 & 2 & 7 \\ 0 & 5 & 7 & 2 \\ 0 & 5 & 7 & 2 \end{pmatrix} \end{array} \qquad t = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \begin{array}{c} X_J = (\{x_1, x_2\} \;\; \{x_3\} \;\; \{x_4\} \;\; \{\overline{x_1}, \overline{x_2}\} \;\; \{\overline{x_3}\} \;\; \{x_4\}) \\ v_J = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & \sqrt{3}/2 & \sqrt{3}/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \sqrt{3}/2 & \sqrt{3}/2 \end{pmatrix} \end{array}.$$

C.9 Function #1910:

The optimal adversary matrix and a matching span program for function #1910 are

$$\begin{array}{c} \\ 0111 \\ 1011 \\ 1101 \\ 1110 \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} 0001 & 0010 & 0100 & 1000 & 0011 & 1100 & 1111 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array} \qquad t = \begin{pmatrix} 1 \end{pmatrix}, \quad \begin{array}{c} X_J = (\{x_1, x_2, x_3, x_4\} \;\; \{\overline{x_1}, \overline{x_2}\} \;\; \{\overline{x_3}, \overline{x_4}\}) \\ v_J = \begin{pmatrix} \sqrt{2} & 1 & 1 \end{pmatrix} \end{array}.$$

C.10 Function #317:

The optimal adversary matrix for function #317, $(x_1 \wedge x_2 \wedge x_3) \vee \mathrm{MAJ}_3(\overline{x_2}, \overline{x_3}, x_4)$, comes from

$$\Gamma = \begin{array}{c} \\ 1000 \\ 0011 \\ 0101 \\ 1110 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} 0111 & 1010 & 1100 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \end{array}$$

A matching span program follows from Lemma 4.12 applied to the span programs for AND_3 and MAJ_3.

C.11 Function #5790:

The optimal adversary matrix and a matching span program for function #5790 are

$$\begin{array}{c} \\ 0000 \\ 0111 \\ 1011 \\ 1100 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} 0011 & 0100 & 1000 & 1111 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \end{array} \qquad t = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad v_J = \begin{array}{c} \begin{array}{cccccc} X_J = (\{x_1\} & \{x_2, \overline{x_3}\} & \{\overline{x_2}, x_3\} & \{\overline{x_1}\} & \{x_2, x_4\} & \{\overline{x_2}, \overline{x_3}\}) \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \end{array}.$$

C.12 Function #385:

The optimal adversary matrix and a matching span program for function #385 are

$$\begin{array}{c} \\ 0000 \\ 1001 \\ 1010 \\ 1100 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} 0111 & 1000 & 1111 \end{array} \\ \begin{pmatrix} \sqrt{2} & \sqrt{2} & 0 \\ 0 & \sqrt{2} & 1 \\ 0 & \sqrt{2} & 1 \\ 0 & \sqrt{2} & 1 \end{pmatrix} \end{array} \qquad t = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad v_J = \begin{array}{c} \begin{array}{ccc} X_J = (\{x_2, x_3, x_4\} & \{x_1\} & \{\overline{x_2}, \overline{x_3}, \overline{x_4}\}) \end{array} \\ \begin{pmatrix} 1 & 0 & \frac{(61+7\sqrt{73})^{1/4}}{2^{3/4}\sqrt{3}} \\ 0 & \frac{(49+5\sqrt{73})^{1/4}}{2^{3/4}\sqrt{3}} & 1 \end{pmatrix} \end{array}.$$

C.13 Function #279, Threshold$_{2\,\text{of}\,4}$:

The optimal adversary matrix for function #279, Threshold$_{2\text{ of }4}$, is

$$\Gamma = \begin{array}{c} \\ 0011 \\ 0101 \\ 0110 \\ 1100 \\ 1010 \\ 1001 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} 0111 & 1011 & 1101 & 1110 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

A matching span program was given in Example 5.1.

C.14 Function #1639:

The optimal adversary matrix and a matching span program for function #1639, equivalent to the formula $(x_1 \wedge x_2) \vee \mathrm{MAJ}_3(\overline{x_1} \wedge \overline{x_2}, x_3, x_4)$, are

$$\begin{array}{c} \\ 0001 \\ 0010 \\ 0111 \\ 1011 \\ 1100 \\ 1101 \\ 1110 \end{array}\!\!\begin{array}{c} \begin{array}{ccccc} 0000 & 0101 & 1001 & 0110 & 1010 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array} \qquad t = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad v_J = \begin{array}{c} \begin{array}{cccc} X_J = (\{x_1, x_2\} & \{\overline{x_1}, \overline{x_2}\} & \{x_3\} & \{x_4\}) \end{array} \\ \begin{pmatrix} 1 & \sqrt{5}/2 & 1/2 & 1/2 \\ 0 & 1 & i & -i \end{pmatrix} \end{array}.$$

This span program is based on the one in Lemma 5.2.

C.15 Function #6014, $\mathrm{EXACT}_{2\,\mathrm{or}\,3\,\mathrm{of}\,4}$:

The optimal adversary matrix and a matching span program for function #6014 are

$$\Gamma = \begin{array}{c} \\ 0011 \\ 0101 \\ 0110 \\ 1100 \\ 1010 \\ 1001 \end{array} \begin{array}{c} \begin{array}{ccccc} 1000 & 0100 & 0010 & 0001 & 1111 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 3^{-1/2} \\ 0 & 1 & 0 & 1 & 3^{-1/2} \\ 0 & 1 & 1 & 0 & 3^{-1/2} \\ 1 & 1 & 0 & 0 & 3^{-1/2} \\ 1 & 0 & 1 & 0 & 3^{-1/2} \\ 1 & 0 & 0 & 1 & 3^{-1/2} \end{pmatrix} \end{array}$$

$$t = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad v_J = \begin{array}{c} X_J = (\{x_1\}\ \{x_1\}\ \{x_2\}\ \{x_2\}\ \{x_3\}\ \{x_3\}\ \{x_4\}\ \{x_4\}\ \{\overline{x_1}\}\ \{\overline{x_2}\}\ \{\overline{x_3}\}\ \{\overline{x_4}\}\,) \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2^{-1/4} & 2^{-1/4} & 2^{-1/4} & 2^{-1/4} \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sqrt{3} & \sqrt{3} & \sqrt{3} & \sqrt{3} \\ 1 & 1 & 1 & -1 & i & -i & i & i & 0 & 0 & 0 & 0 \\ i & -i & i & i & 1 & 1 & 1 & -1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

This span program is based on the expansion $\mathrm{EXACT}_{2\text{ or }3\text{ of }4} = \mathrm{Threshold}_{2\text{ of }4}(x_1, x_2, x_3, x_4) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$, and the span program for $\mathrm{Threshold}_{2\text{ of }4}$ #279 from Example 5.1.

C.16 Function #278, $\mathrm{EXACT}_{1\,\mathrm{of}\,4}$:

The optimal adversary matrix for function #278, $\mathrm{EXACT}_{1\text{ of }4} = \mathrm{Threshold}_{3\text{ of }4}(\overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3 \vee x_4)$, is

$$\Gamma = \begin{array}{c} \\ 0011 \\ 0101 \\ 0110 \\ 1100 \\ 1010 \\ 1001 \\ 1111 \end{array} \begin{array}{c} \begin{array}{cccc} 0111 & 1011 & 1101 & 1110 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

A span program with matching witness size $\sqrt{10}$ follows from Lemma 4.12 applied to the span programs for OR_4 and $\mathrm{Threshold}_{3\text{ of }4}$ #279.

C.17 Function #5736, $\mathrm{EXACT}_{2\,\mathrm{of}\,4}$:

The optimal adversary matrix for function #278, $\mathrm{EXACT}_{1\text{ of }4} = \mathrm{Thr.}_{2\text{ of }4}(x_1, \ldots, x_4) \wedge \mathrm{Thr.}_{2\text{ of }4}(\overline{x_1}, \ldots, \overline{x_4})$, is

$$\Gamma = \begin{array}{c} \\ 0011 \\ 0101 \\ 0110 \\ 1100 \\ 1010 \\ 1001 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1000 & 0100 & 0010 & 0001 & 0111 & 1011 & 1101 & 1110 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

A span program with matching witness size $\sqrt{12}$ follows from Lemma 4.12 applied to the span program for $\mathrm{Threshold}_{2\text{ of }4}$ #279.